Endless minds most beautiful: building open-ended linguistic autotelic agents with deep reinforcement learning and language models

Laetitia Teodorescu

Agents autotéliques linguistiques ouverts avec apprentissage par renforcement profond et modèles de langage

---

Par **Laetitia Teodorescu**

Sous la supervision de Pierre-Yves Oudeyer et Katja Hofmann

Thèse présentée pour obtenir

le grade de Docteur

Université de Bordeaux

École Doctorale de Mathématiques et d'Informatique

Spécialité Informatique

Soumise le 9/10/2023. Défendue le 20/11/2023.

Composition du jury:

Dr. Claire Gardent	Loria, Professeur, CNRS, Université de Lorraine	Présidente
Dr. Daniel Polani	Professeur, School of Computer Science, University of Hertfordshire	Rapporteur
Dr. Todd Gureckis	Professeur, New York University	Rapporteur
Dr. Laura Schulz	Professeur, Brain & Cognitive Sciences, MIT	Examinatrice
Dr. Roberta Raileanu	Research Scienist, FAIR (Meta) London	Examinatrice
Dr. Katja Hofmann	Senior Principal Researcher, Microsoft Research Cambridge	Co-directrice de thèse
Dr. Pierre-Yves Oudeyer	Directeur de Recherche, Inria	Directeur de thése

Résumé de la thèse en français

Cette thèse porte sur les êtres artificiels qui génèrent de la nouveauté intéressante pour toujours. Dans la communauté de machine learning et de calcul évolutionnaire, ce concept s'appelle l'open-endedness (ouverture). Certains chercheurs (notamment en vie artificielle ou Alife) prennent comme exemple de processus ouvert l'évolution naturelle et cherchent a en reproduire les caractéristiques, mais nous sommes intéressés par l'ouverture au niveau du comportement d'agents individuels. Des exemples motivants: un enfant joue et invente de nouvelles règles, de nouvelles aventures pour lui et pour les autres ; un scientifique consacre sa vie à poser de nouvelles questions sur la nature et à trouver la réponse avec des collègues du monde entier ; un peintre rêve d'une nouvelle scène et n'a de cesse de travailler que cette scène n'ait pris vie sur la toile. Ce sont ces comportements qui nous inspirent : comment construire des machines qui s'absorbent dans de telles activités ? Comment pourraient-elles en inventer de nouvelles ? Nous nous situons dans une tradition de recherche qui cherche à répondre à cette question par l'étude de la motivation intrinsèque: quels en sont les mécanismes chez les humains, quels modèles computationnels peut-on en déduire, et comment implémenter ces modèles en utilisant les techniques à l'état de l'art en machine learning aujourd'hui? La motivation intrinsèque a un rôle particulier à jouer dans l'élaboration d'agents ouverts, parce qu'elle sous tend à la fois l'apprentissage chez les enfants et l'innovation chez les adultes. Comme les membres de cette école de recherche, nous nous rendons compte que les êtres artificiels ouverts doivent jouer: non seulement résoudre des problèmes, mais aussi en inventer de nouveaux de manière créative. Dans le prolongement de ces travaux antérieurs, nous étudions l'apprentissage dit autotélique. C'est un paradigme d'apprentissage qui permet à un agent d'inventer ses propres objectifs, d'essayer de les atteindre et de tirer parti de ses échecs et de ses réussites afin d'augmenter ses compétences, et de pouvoir se fixer des buts plus ambitieux.

Cette thèse est centrée sur le langage comme moyen de construire des objectifs créatifs. Le langage est un support parfait pour la créativité : des mots et concepts déjà connus se combinent dans de nouvelles phrases, le langage fait abstraction des détails inutiles, et le langage peut même être le siège de nouveaux concepts. Le langage définit également des espaces intéressants à explorer, comme le savent bien les inventeurs de jeux basés sur le texte, les joueurs de jeux de rôle ou les créateurs d'histoires. Nous adoptons donc la perspective du langage à la fois comme outil cognitif pour générer des buts créatifs dans le cadre d'un apprentissage autotélique mais aussi comme manière de définir un environnement ouvert et complexe. Cette thèse propose des contributions empiriques composées à la fois d'analyses et de nouvelles méthodes permettant de comprendre comment construire des agents linguistiques autotéliques ouverts.

Nous commencerons cette thèse par un chapitre introductif. L'auteur de ces lignes a suivi sa curiosité et écrit une histoire de l'intelligence artificielle, en mettant l'accent sur la succession des

différentes ères. Nous commençons par les origines, des premiers logiciens à l'établissement de l'intelligence artificielle comme discipline scientifique en 1956. Nous discuterons des premiers succès des systèmes logiques et des systèmes jouant au morpion et au dames. Nous passerons sur les espoirs déçus et le premier hiver de l'IA dans les années 70 ou les financements se sont réduits massivement, jusqu'au succès commercial des systèmes experts, le second hiver et la renaissance du connexionnisme qui vit l'élaboration de réseaux de neurones plus complexes. Nous nous attardons ensuite sur les différents moments de l'essor fulgurant de l'IA ces 15 dernières années, avec une emphase sur les techniques que nous utiliserons dans cette thèse, et quelques expositions techniques (notamment lorsque nous introduisons les différents paradigmes d'apprentissage). Le thème récurrent qui émerge de cette histoire récente est que l'essor rapide du machine learning et son succès empirique tiennent en premier lieu a la disponibilité de puissance de calcul et de données er ensuite à l'élaboration d'algorithmes d'apprentissage relativement simples et très flexibles permettant de prendre avantage de cette puissance de calcul et de ces données. Le début de l'ère du deep learning commence avec la victoire écrasante du réseau convolutif AlexNet dans une compétition de reconnaissance d'image en 2012, battant avec une grande marge tous les compétiteurs basés sur des approches de machine learning plus traditionnelles. Nous continuerons avec le début de l'ère de l'apprentissage par renforcement profond (deep RL, de 2015 à 2019) caractérisé par le développement rapide d'agents dans un cadre de prise de décision séquentielle, un paradigme qui a permis de battre les champions du monde de Go, d'échec, et de jeux compétitifs plus complexes encore comme Dota2 ou StarCraft2. Nous finirons ensuite par l'ère la plus récente: celle des grands modèles de langage (LLM) et du passage à grande échelle (scaling, de 2019 à aujourd'hui). La leçon principale de cette nouvelle ère est que passer à l'échelle (pour une architecture similaire: plus de données et plus de paramètres) donnent aux modèles des capacités émergentes qui n'existent pas aux échelles inférieures: nous discuterons rapidement des lois d'échelles ayant été établies récemment caractérisant la performance d'un modèle à son échelle. Nous offrirons ensuite une autre perspective: celle que les LLMs entraînés de manière non-supervisée sur d'immenses corpus ne peuvent dans le meilleur des cas qu'imiter la performance et le raisonnement des humains, mais non aller au delà et inventer leurs propres problèmes comme le ferait un scientifique ou un enfant. Nous inscrivons par là même les travaux sur la motivation intrinsèque et l'apprentissage autotélique dans la continuité des développements récents en IA.

Nous continuons notre introduction par une exposition des phénomènes ouverts (open-ended) les plus saillants dans le monde naturel: l'évolution génétique et culturelle. Après en avoir brièvement exposé les principes, de variation, transmission et sélection, nous nous demandons ce qui peut bien être à l'origine de la variation en évolution culturelle. Nous arrivons à la conclusion que ce sont les mécanismes de la motivation intrinsèque qui sont responsables de cette variation. Nous discutons ensuite de la recherche en psychologie développementale, qui décrit l'évolution des jeunes enfants par une série de stades développementaux, et nous nous établissons ensuite une revue de la recherche en motivation intrinsèque, d'abord chez les humains et ensuite nous regardons les différents modèles computationnels qui ont été proposés. Les modèles computationnels de motivation intrinsèques se divisent en trois catégories: les modèles basés sur la connaissance (ou une motivation est générée en fonction de la relation entre un stimulus/une observation et les modèles internes d'un agent), les modèles basés sur la compétence (ou une motivation est générée en fonction de la compétence d'un agent sur un but donné), et les modèles basés sur la structure du flot sensorimoteur (l'empowerment ou les récompenses liées à la régularité tombent dans ce cadre). Un des avantages des motivations intrinsèques basées sur la compétence sont la relation de ces récompenses internes avec le répertoire de compétence de l'agent. Nous développons ces intuitions en présentant ensuite le

paradigme de l'apprentissage autotélique, ou un agent sélectionne ses propres buts et est récompensé pour essayer de les accomplir. Nous discutons brièvement des travaux existants dans ce cadre et des avantages de cette forme d'apprentissage intrinsèquement motivée pour la construction de répertoires de compétence ouvert. Nous nous attardons sur les processus de sélections de buts, et nous remarquons que sélectionner des buts en fonction de compétence intermédiaires (dans des environnements déterministes) et le progrès d'apprentissage (plus généralement) permet de séquencer des phases développementales selon une structure similaire à ce qui est observé chez les jeunes enfants. Nous disons un mot de la sélection de buts appropriés comme une version d'apprentissage par curriculum automatique (automated curriculum learning: ACL). Nous concluons ensuite sur la présentation d'un travail séminal antérieur utilisant des buts linguistiques, et utilisant les propriétés de productivité et de compositionnalité du langage comme mécanisme d'imagination de buts nouveaux.

Notre premier chapitre expérimental porte sur les agents autotéliques dans les jeux basés sur le texte. Comme dans l'introduction, nous avons pris le temps de présenter un contexte général, cette fois-ci sur les théories du langage en psychologie cognitive et en linguistique, en essayant de dégager ce que nous savons sur ce qu'est un symbole, et sur le caractère ouvert (open-ended) du langage. Nous nous arrêterons notamment sur certaines des théories linguistiques tentant de caractériser cette nature ouverte. Nous passons ensuite à notre première contribution expérimentale où nous présentons une étude d'un agent autotélique utilisant des objectifs linguistiques dans un environnement textuel complexe, ScienceWorld. Un environnement textuel est un environnement d'apprentissage par renforcement ou les observations et les actions sont du texte. L'environnement Scienceworld contient des règles de physique et thermodynamique élémentaire. Nous nous basons sur ces propriétés pour définir une hiérarchie d'objectifs linguistiques que l'agent peut maîtriser, et l'apprentissage de l'association des objectifs et du comportement se fait grâce au feedback d'un partenaire social (selon le principe du hindsight expérience relabelling, ou un comportement effectué est considéré comme une démonstration du but effectivement accompli, et non du but originellement visé). Dans l'étude par ablations menée, nous établissons que la sélection de buts basée sur la compétence intermédiaire est déterminante dans l'apprentissage de la maîtrise de la hiérarchie de buts, tout comme l'est est un retour d'information filtré du partenaire social. Ce dernier point est lié à la difficulté d'entraîner un modèle massivement multitâche avec des modèles standard d'apprentissage par renforcement profond, à partir de zéro. Nous passons ensuite à notre deuxième étude, qui présente une nouvelle méthode intitulée LMA3 (Language Model Augmented Autotelic Agents), une méthode intégrant de grands modèles de langage (LLM) dans l'architecture autotélique pour la génération d'objectifs ouverts. Nous instantions notre agent dans un environnement textuel simple (CookingWorld), et nous utilisons le fait que les trajectoires de l'agent se déroulent en langage pour implémenter le module de sélection de buts, la fonction de récompense conditionnée par le but et le module de labellisation, par des appels à des modèles de langage (ChatGPT dans nos expériences). La sélection de buts peut donc générer des buts ouverts, la fonction de récompense calculer des achèvements de buts pour des buts arbitraires, et la fonction de relabellisation, jouant dans ce travail le meme role que le partenaire social du travail précédent, relabelliser des comportements arbitraires tout en émettant des buts rétrospectifs pertinents. Nous démontrons que notre agent est capable de découvrir des milliers de buts meme dans cet environnement relativement simple. Nous établissons par avance une batterie de buts d'évaluation dont nous pouvons mesurer programmatiquement la complétion, et en évaluant LMA3 sur ces buts de test nous démontrons qu'il a été capable de les découvrir au cours de son exploration. Nous mesurons également le nombre et la diversité des buts atteints au fur et à mesure de l'exploration pour différentes ablations de notre système. Cette analyse met en valeur l'importance de formulation de la prompt (les instructions au LLM), notamment l'importance de conseils sur ce qui

constitue un but important pour la génération de but et la relabellisation. Nous concluons ce chapitre en soulignant les promesses des grands modèles pour l'apprentissage autotélique, en particulier dans des environnements plus ouverts.

Notre deuxième chapitre expérimental fournit des éléments pour aborder le problème de l'ancrage du langage (language grounding) dans les agents autotéliques. Quels pourraient être les bons principes pour qu'un agent relie ses objectifs linguistiques à ses observations sensorimotrices ? Après une introduction sur l'ancrage, les arguments pour se concentrer sur la fonction de récompense et les architectures basées sur l'objet telles que les réseaux neuronaux graphiques (GNN), nous passons à notre troisième contribution. Dans cette étude, nous présentons SpatialSim, un ensemble de tâches d'identification et de comparaison de configurations spatiales d'objets. Nous testons plusieurs architectures de réseaux neuronaux basées sur les GNN ainsi que de puissantes architectures de vision par ordinateur et nous concluons que, sur une série de mesures, les architectures présentant le calcul le plus relationnel sont les plus performantes. Dans notre troisième étude, nous nous demandons quels principes de conception architecturale pourraient être nécessaires pour fonder un langage avec une sémantique spatiale et temporelle. Nous définissons un langage artificiel capable de décrire des relations spatiales, des relations temporelles et des prédicats à extension temporelle, et nous essayons d'apprendre une correspondance (ou une fonction de récompense) prédisant si une trajectoire donnée d'un agent correspond à une phrase donnée. Nous constatons que les architectures relationnelles comportant le moins d'hypothèses préalables sont les plus performantes. Nous concluons ce chapitre par une vue d'ensemble de ce qui pourrait être nécessaire pour fonder des objectifs linguistiques réellement ouverts, comme ceux que nous visons à étudier dans cette thèse.

Dans notre dernier chapitre expérimental, nous développons une idée qui apporte des éléments de réponse à la fois au sujet de la complexité des environnements soulevés dans le premier chapitre expérimental et au sujet de l'ancrage du langage: implémenter des agents autotéliques dans des environnements de programmation. Les possibilités de la programmation sont infinies et les buts peuvent être exprimés pour les agents autotéliques sous forme de tests unitaires. L'exécution d'un programme (avec ou sans erreur) fournit une vérité de base par rapport à laquelle l'agent peut s'entraîner, ce qui nous permet de contourner le problème de l'ancrage par apprentissage de fonctions de récompense. Sur la base de cette idée, nous proposons deux contributions. La première introduit ACES (Autotelic Code Exploration with Semantic descriptors), une méthode pour générer une diversité de puzzles de programmation. Cette méthode, proche d'une version autotélique d'un algorithme de qualité-diversité, est basée sur la définition d'un espace formé par des descripteurs sémantiques. Les méthodes de qualité-diversité (qui sont des algorithmes de population ou le but est d'évoluer un ensemble d'individus à la fois divers et localement optimaux) comme map-elites ont besoin d'un espace dans lequel caractériser la nouveauté d'un individu. Ces espaces sont souvent faits main ou basés sur un plongement dans un espace latent appris par un modèle, mais ces solutions ont des problèmes (respectivement le manque de flexibilité et la malédiction de la dimensionnalité). Pour définir notre espace de caractérisation sémantique nous établissons une ontologie de 10 compétences de programmation (mathématiques, récursivité, problèmes de graphes...) et nous catégorisons les puzzles de programmation générés par un LLM dans l'espace 10-dimensionnel composé des booléens d'appartenance aux différentes compétences de programmation que ce puzzle requiert ou non. Cette catégorisation est aussi effectuée par le même LLM. Nous montrons que l'utilisation de cet espace sémantique combinatoire conduit à la génération d'une plus grande diversité de puzzles, sur une variété de mesures. Nous passons à la deuxième contribution de ce chapitre, où nous présentons l'apprentissage autotélique dans le domaine des puzzles de programmation comme un jeu entre deux agents : un "Setter" générant des puzzles difficiles, mais encore résolubles, et un "Solver" apprenant à

les résoudre. Nous appelons ce jeu Codeplay. Dans de premières expériences, nous montrons que le Setter, un modèle de langage ajusté avec l'apprentissage par renforcement, parvient à générer des puzzles qui sont difficiles, solubles, et relativement nouveaux. De plus, nous arrivons à contrôler le degré de nouveauté ou la contrainte de difficulté des puzzles générés par le Setter en ajustant deux composants de la fonction de récompense du Setter. Par ces expériences préliminaires, nous démontrons ainsi une première étape pour l'implémentation de l'algorithme Codeplay final.

Acknowledgements

I would like to thank first and foremost my advisors; Pierre-Yves Oudeyer, for trusting me and giving me guided freedom to explore, and Katja Hofmann, for her mentoring and unwavering commitment to getting things done.

I have been lucky to perform this research alongside wonderful collaborators. I thank Tristan Karch for his energy and liveliness in collaboration. I thank Guillermo Valle for teaching me about normalizing flows. I wholeheartedly thank Marc-Alexandre Côté and Eric Yuan for welcoming me at Microsoft Research Montréal, for their help with text worlds and for access to GPT3 to perform the LMA3 experiments. My greatest thanks to Cédric Colas, for innumerable discussions, wonderful projects, and continued friendship. I have had the pleasure of supervising Julien Pourcel, who did amazing work during his master's internship. I thank Matthew Bowers, Patrick Haluptzok and Adam Tauman Kalai for believing in my ideas on autotelic language models.

I am grateful to all the people who helped me co-organize the Language and Reinforcement Learning workshop at NeurIPS 2022. This was an adventure and would not have been possible without Paul Barde, Tristan Karch, Cédric Colas, Pratysha Sharma, A.P. Jacob, Laura Ruis and Jelena Luketina. I am similarly thankful for my co-organizers for the upcoming Intrinsically-Motivated Open-Ended Learning workshop at NeurIPS 2023: Cédric Colas, Cansu Sancaktar, Junyi Chu and Nadia Ady.

The Flowers lab at Inria Bordeaux is a rare research environment, both pluridisciplinary and cutting-edge, and home to a unique school of thought. Particular thanks go to Nathalie Robin for making the Montréal stay possible. I would like to thank all my current and former colleagues for scientific or technical discussions: Thomas Carta, Cément Romac, Rémy Portelas, Alexandre Péré, Cément Moulin-Frier, Héléne Sauzéon, Cécile Mazon, Mayalen Etcheverry, Gautier Hamon, Maxime Adolphe, Matisse Poupard, Isabeau Saint-Supéry, Rania Abdelghani. Alex Ten, Nicolas Yax, Marion Pech and Grgur Kovac. Special thanks go to Eleni Nisioti for a friendship that I hope will endure for long.

All my thanks go to my family, for believing in me; my mother, my dad, Ivan and Stefania; my brothers Luca and Julian, and my sisters Iulia and Smaranda. It would be too long to thank all the friends who have supported me these four years, but I am blessed to have them at my side. I thank Teven for the drive, and Matt for the unwavering love and support, oceans away.

Preamble

This is a thesis about artificial beings that endlessly generate interesting novelty for ever. A child plays and invents new rules, new adventures for herself and for others; a scientist devotes her life to asking new questions about Nature and finding the answer with colleagues from all over the world; a painter dreams of a new scene and has no rest until it has come alive on the canvas. These are the kind of behavior that inspire us: how could we build machines that absorb themselves in such activities? How could they invent new ones? We walk in the footsteps of numerous others before us (to cite only a few: Oudeyer & Kaplan (2007b); Baranes & Oudeyer (2013c); Schmidhuber (2013); Colas et al. (2020b); Forestier et al. (2022b); Colas et al. (2022c)). As they did, we realize that artificial playful beings need to not only solve problems but *creatively invent new ones*. After Csikszentmihalyi (1998), Steels (2004) and Colas et al. (2022c), we investigate **autotelic learning**, whence an agent invents its own goals, tries to achieve them and leans from its failures and successes.

This thesis is centered around **language as a way to build creative goals**. Language is a perfect medium for creativity: old words combine in new phrases, words abstract away unnecessary details, and language can even be the seat of novel concepts. Language also defines interesting spaces to explore: as the inventors of text-based games, the players of role-playing games, or the builders of stories know well. We believe that language holds the key to open-ended learning: learning new things forever without converging.

Introduction We will begin (Chapter 1) by an introductory chapter. The author of these lines has followed their curiosity and written a history of artificial intelligence (AI: Section 1.1), emphasizing the different eras, from the early days of logic to connexionism, to the beginning of deep learning, to deep reinforcement learning and to the current era of scaling and language models. Some sections (such as 1.1.3 or 1.1.5) delve into technicalities and can serve as background for some of the techniques used in our experimental contributions, but are not stricly necessary to understand the main results of this work. We continue the Introduction with an overview (Section 1.2) of known open-ended systems, such as natural and cultural evolution (Section 1.2.1 and 1.2.2). We then move on (Section 1.2.3) to review developmental psychology and intrinsic motivation in humans, culminating with a detailed overview of computational models of intrinsic motivation (Section 1.2.3) and autotelic AI (Section 1.2.3). We conclude the introduction with an introductory summary of the contributions (Section 1.3.1, and Section 1.3.2 for the list).

Linguistic autotelic agents Our first experimental chapter (Chapter 2) will be about autotelic agents in text-based games. As in the introduction, the author has taken some time to provide a general background (Section 2.1), this time on theories of language in cognitive psychology and

linguistics, trying to tease out what we know about the open-endedness of language. We then move on (Section 2.2) where we present a study of an autotelic agent using language goals in a challenging text-based game, where observations and actions are linguistic. We define a hierarchy of language goals the agent can master, and learning to associate goals and behavior happens through the feedback of a social partner. In ablation studied we establish that goal-selection based on intermediate competence is instrumental in learning to master the goal hierarchy, as is filtered feedback from the social partner. We then move on to our second study (Section 2.3) presenting LMA3, a method integrating Large Language Models (LLMs) into the autotelic architecture for open-ended goal generation. We demonstrate that our agent is able to discover thousands of goals and measure the diversity of these goals in various ablations. We conclude this chapter by outlining the promises of large models for autotelic learning, especially in more open-ended environments.

Object-based representations for language grounding Our second experimental chapter (Chapter 3) provides some elements for tackling the language grounding problem. What could be good principles for an agent to link its language goals to its sensorimotor observations? After an introduction on grounding, arguments for focusing on the reward function, and object-based architectures such as Graph Neural Networks (GNNs: Kipf & Welling (2016); Battaglia et al. (2018a)) (Section 3.1), we move on to our third contribution (Section 3.2.2). In this study we introduce SpatialSim, an ensemble of tasks of identification and comparison of spatial configurations of objects. We test several neural network architectures based on GNNs as well as powerful computer vision architectures and conclude that on a series of measures the architectures exhibiting the most relational computation perform best. In our third study (Section 3.3), we ask ourselves what architectural design principles could be needed for grounding language with spatial and temporal semantics. We define an artificial language with the power to describe spatial relations, temporal relations, and temporally-extended predicates, and try to learn a correspondence (or reward function) predicting if a given trajectory of an agent corresponds to the the given sentence. We identify that relational architectures with the least amount of prior assumptions perform best. We then conclude this chapter by an overview of what could be needed to ground truly open-ended language goals, as the ones we aim to study in this thesis.

Code-generating agents In our final experimental chapter (Chapter 4), we develop an insight that solves both the issues of working with open-ended environments and of grounding. The possibilities of programming are endless, and goals can be expressed for autotelic agents as unit tests. Programs are idealized language. Program execution provides a ground truth against which the agent can train, allowing us to sidestep the problem of grounding by learning reward functions (as in Chapter 3). Building on this insight we propose two contributions. The first (Section 4.2) introduces ACES, a method for generating a diversity of programming puzzles, based on a categorization of programming puzzles as involving a particular combination of programming skills. Both generation of new puzzles and the prediction of their category is done with an LLM. We show that the use of this combinatorial semantic space leads to the generation of a larger diversity of puzzles, on a variety of measures. We move on to our second contribution of the chapter (Section 4.3), where we cast autotelic learning in the domain of programming puzzles as a game between two agents: a Setter generating hard, but still solvable, puzzles, and a Solver learning to solve them. We call this game Codeplay. In first experiments, we show that the Setter, a smaller language model finetuned with reinforcement learning, manages to generate puzzles that are hard, solvable, and relatively novel: this is an important early result to the implementation of the full Codeplay algorithm. We conclude by discussing the numerous avenues for further work these contributions open.

Contents

Chapter 1

Introduction

Preamble to the introduction

Building autonomous machines that open-endedly explore their environment has been a long-standing goal of artificial intelligence research. In this thesis, our goal is to contribute to this tradition. To ground our investigation we begin by reviewing both the history of AI up to the latest state-of-the-art techniques with an emphasis on techniques that have achieved major milestones and that will be relevant for our contributions, as well as a discussion of the tradition within which we situate ourselves: intrinsically-motivated open-ended learning (Meyer & Wilson, 1991; Baldassarre & Mirolli, 2012; Der & Martius, 2012; Forestier et al., 2022b; Schmidhuber).

Thus, this introduction chapter first takes a dive in the history of AI (**Section 2**). We begin with the very first attempts at making thinking machines, through the links between AI and cognitive science, and to the emergence of modern AI techniques based on machine learning 15 years ago. We spend some time detailing the most important techniques used today, from deep reinforcement learning (Mnih et al., 2015) to Transformers (Vaswani et al., 2017) and large language models (Brown et al., 2020); we spend time in particular on techniques used in this thesis. Thus, this discussion also serves as technical background on the experimental contributions made in this paper. We will end this review with a remark: AI is usually understood as the set of techniques to automatically solve problems. How to use them to invent new ones?

We will then (**Section 3**) discuss open-endedness (Wang et al., 2019; Stanley & Lehman, 2015) and intrinsic motivation (Oudeyer & Kaplan, 2007b) as inspiration sources in our quest to create systems that endlessly generate novel problems, novel solutions. We will begin with a discussion of natural evolution, the most compelling open-ended process that we can observe, forever generating novel species, and emphasize the role of feedback loops between environment and organism (Odling-Smee et al., 2003). We will point out that cultural evolution can be understood in similar ways (Laland & O'Brien, 2011). We will continue by arguing that the driver at the heart of cultural evolution and innovation are intrinsic motivation and play (Oudeyer & Kaplan, 2007b; Chu & Schulz, 2020). We will accordingly devote the finishing sections of this introduction by presenting the different approaches and theories surrounding intrinsic motivation, in humans and machines.

We will then end (**Section 4**) with a presentation of each of the following content chapters and a list of our contributions.

1.1 An overview of Artificial Intelligence

Artificial Intelligence is a long-standing dream that has its roots before even the first computers, with philosophers', logicians' and psychologists' inquiries into the nature of reason and the mind. However, this dream started started to solidify in the course of the second half of the 20th century. The definition of intelligence itself is elusive and might depend on the author (Legg et al., 2007); it is one of these things that we easily recognize but have a hard time quantifying, which is problematic for a field of study seeking to reproduce it. Some define AI as the science of automated problem solving, or as the study of how to build rational agents (Russell & Norvig, 2020). Interestingly, these definitions leave to the research community the decision of which problems to solve or which objectives the rational agent should attain.

A distinction has emerged over time between cognitive science, studying the operation of the mind in humans through the analogy between human thought processes and the workings of a computer, and AI, which is unconstrained by trying to provide a scientific account of the human mind. Nevertheless, *Homo Sapiens* is the only example of flexible reasoning and decision making and the study of human cognition regularly informs intuitions about building automated intelligent systems. With the success of recent compute-heavy machine learning algorithms in particular (LeCun et al., 2015), giving rise to adaptable AI systems, AI researchers regularly turn to human capabilities as inspiration for what capabilities and tasks AI should solve. This is easily seen through claims of human-level performance (Mnih et al., 2015), human-timescale adaptation (Team et al., 2023) and similar wordings in AI breakthroughs. More generally, modern machine learning researchers and engineers often claim Artificial General Intelligence as their goal (among many others Clune (2019); Altman (2023)); the concept is ill-defined but it usually is taken to designate systems that have human or superhuman performance in a wide range of (economically relevant) skills.

In this section, we present a short history of the field of AI, outlining the major milestones and paradigm shifts and what lead to them. After giving an account of the historical roots of AI we will focus in more detail on the techniques used in a contemporary machine learning context, supported by developments in the past 15 years or so. This will be an opinionated history, guided by wider significance (milestones, societal impact or impact in the other sciences) and by concepts that are specifically relevant to this thesis. This will also serve as background for all the techniques used in this manuscript. Once this overview is done, we will argue for the importance of *Developmental Artificial Intelligence*, the branch of AI studying how to solve automatically the types of challenges encountered by developing humans. We believe that solving some of these problems could lead to progress in general AI systems, and will defend this idea at the end of the section.

1.1.1 The origins

Philosophers have wondered for millennia about the human mind and its capabilities, and old myths abound with stories of artificial beings: Galatea, Pygmalion's statue, or the Golem, to name a few. The mind seems like a unique phenomenon in Nature, and to explain it most philosophers in the Western tradition have posited that it is made of a different essence than that of ordinary matter: this position is called dualism. This worked well within the various spiritual frameworks that were dominant in the Western world (such as Christianity from the 5th century to the Industrial Revolution). Monism is the opposite belief: that mind and matter are actually made of the same essence. Some form of monism is a prerequisite for the belief that AI is possible: since intelligence is a part of the

Figure 1.1: Rosenblatt's perceptron was implemented on analog machines.

human mind, believing that mind and matter are completely separate leads to considering that the effort to build intelligent machines is a fruitless one. Dualism saw its first cracks with the philosophy of Spinoza, and around the same time Leibniz started designing one of the oldest computing devices, the stepped reckoner. He additionally crafted plans to create a universal language for thought and reasoning, fully composititonal and non-ambiguous, composed of symbols and of the logical operations to manipulate those symbols, which he termed the *Universal Characteristic*. He believed that scholars using this language could simply compute the result of any reasoning process, and that debate among scientists and philosophers would be greatly simplified. The *Universal Characteristic* can be seen as a precursor to the program of AI. A similar research endeavor motivated the logicians of the late 19th century and early 20th century, such as Gottlob Frege, one of the founding figures or analytical philosophy. He seeked to ground all of mathematics into logic (a program that was carried to term by Russell and Whitehead in their 1923 *Principia Mathematica* (Whitehead & Russell, 1923)) and invented the first predicate calculus, laying the groundwork for all further developments in logic and formal systems.

In the first half of the 20th century, the discovery that the operations of the brain were carried out by a dense network of individual neurons firing discrete signals led to the development of computational models of neurons and neural networks, most notably by McCulloch and Pitts (McCulloch & Pitts, 1943). Norbert Weiner started developing cybernetics (Wiener, 1961), his theory of self-regulating systems, shortly after the development of theories of universal computation by Turing (Turing, 1937) and Church (Church, 1932), as well as after the theory of information of Shannon (Shannon, 1948) gained influence. This fertile intellectual ground led a group of researchers in computer science, mathematics and logics to convene in a workshop in Dartmouth in 1956. There, the participants expressed their confidence in the fact that that implementing the operations of mind inside a computer was possible, and the field of AI was officially born and its program defined. Early optimism led to a flurry of developments such as game-playing AI (see the checkers-playing program of Samuel (1959)), language understanding and conversational programs (for instance Joseph Wiezenbaum's ELIZA (Agassi & Wiezenbaum, 1976)), as well as formal theorem provers, robotics and vision (Newell et al., 1959; Winograd, 1971). The era also saw the development of Rosenblatt's perceptron (Rosenblatt, 1957), a working version of McCullch and Pitt's mathematical neuron. It was possible to train this neuron, by repeated exposure to input-output pairs, to approximate the function between inputs and outputs: the first machine learning program was born.

However this early enthusiasm did not last, as the early pioneers of AI failed to deliver in the early 70s the grandiose achievements they had promised in the prior decade. Funding dried up and the first AI winter began around 1974. Among interesting criticism of AI approaches of the time were those of Hans Moravec. His well-known paradox states that what is hard for us is easy for machines and vice versa. He meat that logic and multistep reasoning are comparatively easy to implement in a computer but the more mundane tasks such as object recognition, keeping one's balance or running up the stairs were wildly out of reach for AI systems of the time. His critique of early AI efforts is that the computers of the time were massively underscaled, by a factor of one million approximatively according to his estimation (Moravec, 1976). The metaphor he used was that at sufficient speed, anything could fly; but the quest to build artificial flight was doomed until enough horsepower could be harnessed in airplanes. The publication of Minsky and Papert's book Perceptrons (Minsky & Papert, 1969), where severe limitations on the classes of images a single-layer perceptron could distinguish, was a decisive blow to the study of neural networks.

A rebirth of AI in the 80s happened due to the initial commercial success of expert systems, logical systems incorporating all known facts of a specialized domain in to a knowledge graph. After generating a great amount of initial excitement and some interesting such as programs for automated mathematical discovery such as Automated Mathematician (Lenat, 1976), Eurisko (Lenat, 1983) and the knowledge base Cyc (Lenat, 1995), this wave of AI faded too and another AI winter started in the late 80s and continued up to the late 90s. Within this context, the invention of neural net training through the backpropagation of errors (Rumelhart et al., 1986a) and the publication of the widely influential Parallel and Distributed Processing book (Rumelhart et al., 1986b) revived research into neural networks. Backpropagation made possible the training of arbitrary layers of neurons connected by a wide variety of mathematical operators. Thus, more complex architectures could be invented and trained, and by the late 90s researchers had already invented recurrent neural networks (RNNs: Elman (1990)), convolutional neural networks (CNNs: Lecun & Bengio (1995)), and long-term-short-memory (LSTMs: Hochreiter & Schmidhuber (1997)) units. More in-depth accounts of neural network techniques will be done in the following sections, but the general story after the rebirth of connectionism in the late 80s has been the one of the success of machine learning in general. The availability of cheap computing power due to Moore's law, of exponentially increasing available data due to the emergence of the Internet, of efficient and flexible neural network architectures and training paradigms, and to increasingly usable and powerful software stacks (Abadi et al., 2015; Bradbury et al., 2018; Paszke et al., 2019) led to the widespread successes of AI in the second decade of the 21st century. Looking back at this rapid evolution, Richard Sutton formulated the *bitter lesson* (Sutton, 2019): approaches to AI that can leverage the increasing amount of compute win in the long-term. Researchers may try to formalize their understanding of a domain, such as language or vision, and input it into a machine, and this might lead to improvements in the sort term. However, these approaches will not scale compared with general methods based on search and learning. This conclusion mirrors the one of Hans Moravec 43 years earlier.

1.1.2 The deep learning revolution, supervised and self-supervised learning

Machine learning is the science of making algorithms that improve on tasks based on experience. Most of contemporary machine learning is heavily reliant on mathematical optimization. Performance of an algorithm is usually defined through a metric (cost function, reward, loss), and the algorithm's responses to input is controlled by a set of parameters that will vary over the course of training to optimize this metric, either maximizing or minimizing depending on the problem. The predictive

Figure 1.2: Deep Neural Nets work by building increasingly complex representations, or *features*, of their data. Olah et al. (2017) developed a technique for visualizing those features, allowing us to gain insight on the dimensions of the data that influences a model's prediction. The top row represents a class of images, the bottom row a visualization of a perfect exemplar of the class according to a CNN. Figure from Olah et al. (2017).

algorithm is called a *model*, and the computational relationship between inputs, parameters, and outputs is called the *architecture* of the model. Deep learning (LeCun et al., 2015) is the use of neural networks in a machine learning architecture.

There are 3 main categories of machine learning paradigms:

- **Unsupervised learning**: this paradigm assumes we have some unlabeled data and that we aim to learn something about its underlying structure. Common examples include dimensionality reduction or clustering.
- **Supervised learning**: in this setup we assume that we have access to some samples of a distribution of examples and labels. The goal is to learn to predict labels on new samples as accurately as possible. An interesting paradigm that has emerged recently is **Self-Supervised learning**. It uses the methods of supervised learning to learn in settings close to the ones of unsupervised learning, namely, considering that the data itself or transformation thereof can be used as labels to learn from inputs only. Because these methods are applicable to massive corpora of unlabeled data they have been responsible for some of the most recent successes in machine learning.
- **Reinforcement learning**: this paradigm concerns itself with the behavior of simulated agents, acting in an environment in order to maximize a scalar signal known as a reward. Intuitively, this corresponds to learning to act in a sequential decision-making process via trial and error. This setup is more involved than supervised learning but is more powerful in modelling a wide range of problems an autonomous artificial system might encounter.

In this section we will cover generalities and major milestones in supervised and self-supervised learning learning. Reinforcement learning will be covered in the following section.

In supervised learning we are faced with a joint probability distribution over two random variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, $p(X, Y)$. We are given a functional hypothesis space $\mathcal{H}$ composed of function from $\mathcal{X}$ to $\mathcal{Y}$ and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The goal of supervised learning is to find the function in the hypothesis space that minimizes the loss $\ell(f(X), Y), f \in \mathcal{H}, (X, Y) \sim p$. In other words, the supervised learning problem is expressed as the following optimization problem:

$$f^* = \arg\min_{f \in \mathcal{H}} \int_{(X,Y)} \ell\big(f(X), Y\big)\, p(X, Y)\, \mathrm{d}X\, \mathrm{d}Y \tag{1.1}$$

$\mathcal{H}$ is usually given by a family of functions parametrized by $\theta \in \mathbb{R}^d$. Since we cannot access the data distribution directly we estimate the loss over our finite dataset $\mathcal{D}$ of (x, y) samples and minimize that function with respect to the parameters. This procedure is known as *empirical risk minimization*:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \sum_{(x,y) \in \mathcal{D}} \ell\big(f_\theta(x), y\big) \tag{1.2}$$

The most standard loss functions used are the mean-squared error (MSE) for regression tasks, when $\mathcal{Y} = \mathbb{R}^n$, and the binary cross-entropy (BCE) for binary classification tasks, when $\mathcal{Y} = [0..1]^n$. Since d is usually high and the loss is usually non-convex in most real-scale machine learning settings the optimization procedure above is usually performed by some form of stochastic gradient descent (Bishop, 1999) over minibatches randomly drawn from the training dataset, and evaluated on the held-out test dataset. Pre-deep learning era machine learning methods usually used parametrized linear functions over a feature space derived from $\mathcal{X}$. These features were usually by studying the domain of interest and extracting quantities of interest by a manual mathematical procedure (for instance SIFT features in computer vision or MFCC features in speech recognition (see discussion in the introduction of Goodfellow et al. (2016))). More complex function spaces involve non-linear parametric function composition of which neural networks are a special case, since neural networks are in their most classical form a collection of layers of linear functions sandwitched between non-linearities such as the sigmoid or hyperbolic tangent functions. Neural nets were originally difficult to train because of vanishing or exploding gradients (Glorot & Bengio, 2010; Pascanu et al., 2013). Indeed, when performing the stochastic gradient descent procedure, the gradient of the loss with respect to the function parameters has to be computed, and since the invention of the backpropagation algorithm (or reverse-mode differentiation) the gradient of one layer of a neural net is computed from the gradient of the layer before it. Propagating the gradient backward through the sigmoid or tanh funtions shrinks it dramatically, so training nets with more than 2 layers was very hard in the early days.

The modern era of machine learning started when a deep CNN called AlexNet (Krizhevsky et al., 2012) won the large-scale image recognition challenge on the ImageNet (Russakovsky et al., 2015) dataset by a very wide margin. The critical element of this success was the depth of the network, for which rectified layer units (ReLU), rather than tanh, were used, greatly speeding up training and improving performance. This simple modification eliminated most of the vanishing gradient problem and made truly deep architectures feasible. Another crucial aspect of the work was technical: the authors wrote their code so as to be executable on the graphical processing units (GPUs) originally developed for the parallel mathematical processing required by video games as other computer graphics applications. This allowed Krizhevsky and colleagues to muster the computing power to train their network. This is an important point (remember Moravec's remarks and Sutton's bitter

lesson discussed in the preceding section) and compute and scale effects will be recurrent themes in the recent history of AI.

Deep learning had started: and a Cambrian explosion of neural architectures started being applied to many supervised tasks, from image recognition (Szegedy et al., 2015; He et al., 2016b) to segmentation to point cloud tasks in computer vision, to applications in speech recognition, to sequence-to-sequence (seq2seq) tasks like machine translation (Sutskever et al., 2014; Bahdanau et al., 2014). Supporting these developments are decreasing costs of graphical processors, developments in general purpose stabilizing measures such as dropout (Hinton et al., 2012) to avoid overfitting, batch or layer normalization to smooth gradient updates and make the optimization process better behaved (Ioffe & Szegedy, 2015; Ba et al., 2016), or new optimizers (Kingma & Ba, 2014), the appearance of software stacks to flexibly implement new algorithms (Abadi et al., 2015; Bradbury et al., 2018; Paszke et al., 2019), and the multiplication of benchmarks and rich datasets for an increasing amount of tasks.

Sequence models will be of particular interest to us for the work presented in this manuscript so let us review the first recurrent models shortly before spending a bit more time on Transformers (Vaswani et al., 2017) in a dedicated section. Sequence models refer to neural architectures that operate on sequences. A typical task would be to classify text, where the input strings would be considered a sequence of characters. Other classical tasks are seq2seq tasks, where after being given a sequence input the goal is to predict a sequence output, such as in machine translation. After the seminal work of Elman on Recurrent Neural Networks (RNNs) (Elman, 1990) and Hochreiter and Schmidhuber's invention of the long-short-term-memory unit (Hochreiter & Schmidhuber, 1997) – whose additional cell channel was designed to maintain information over long timespans, the issue of exploding or vanishing gradients prevented the application of these recurrent architectures on long sequences (Glorot & Bengio, 2010). Indeed, recurrent architectures are trained with the backpropagation through time algorithm, unrolling the model through time and passing the gradient though all the temporal replicas of the recurrent unit. Even when stabilizing methods had been found for mitigating training issues, the networks had trouble computing long-range dependencies on text. The introduction of attention methods in RNNs mitigated the issue by allowing direct long-range information flow bypassing the sequential processing characteristic of recurrent nets (Bahdanau et al., 2014). In the context of machine learning, attention is a set of scalars computed between a vector x and vectors y_i. x and the y_is are projected to a common space by matrices W_x and W_y and the attention is computed as the softmax between the sequence resulting from the scalar product of $W_x x$ and each of the $W_y y_i$s. The attention is computed, in the context of sequence tasks, between the current element of the sequence and all those that came before, potentially a long distance away.

Self-supervised learning As mentioned above, self-supervised learning is unsupervised learning with the methods of supervised learning. In most cases, the data itself has enough internal structure to be able to provide sufficient supervision, either for learning useful representations of the data that can be used in downstream tasks, or for learning a generative model of the data. A first important method in self-supervised learning is **contrastive learning**. This paradigm relies on knowledge that certain points in the dataset are similar to each other and that others are different, in some arbitrary way (for instance one can obtain close points by simple transformations). Contrastive learning aims to map (through a learnable function/neural network encoder) the high-dimensional input points to a lower dimensional embedding space where the points that share the arbitrary similarity in the input space are close together and the points that do not are far apart. The seminal work of Hadsell et al. (2006) compare this to a mass-spring system. This method has successfully been applied to

representation learning in image domains (Chen et al., 2020b). In that work, unlabeled images are subjected to a combination of transformations (rotation, color change, ...). The images sharing the same source are brought together and the ones sharing a different source are pulled apart. The method learns representations that can be decoded by linear layers to predict image classes, because by learning under the contrastive method, general information about the images has been learned, and this information is usefully adapted by finetuning (further training on a smaller dataset) to downstream, supervised tasks.

A second important type of self-supervised technique uses sequence modelling to factorize the distribution of datapoints into a product of conditional probability. In short, this means taking a sequence and modelling its probability as the product of the probability of the first element, the second knowing the first, the third knowing the first two, etc. Training such a factorized model entails taking a sequence of ordered elements $(s_i)_{i \leq n}$ and trying to predict the n + 1th element: the loss is the negative log-likelihood of the sample s_{i+1} under the model conditioned on $(s_i)_{i \leq n}$. This loss can be estimated for each element in the sequence. Once the probability model has been trained, it also serves as a generative model of the data: one can decode from the prediction of s_i a candidate value (for instance if the model models a probability distribution one can sample a value from this distribution), and this value is appended to the existing sequence and can serve as conditioning for further prediction. This is called autoregressive generation (because the model performs regression on the basis of elements it has generated itself). While various tasks can be cast as sequence modelling (such as pixel prediction (Chen et al., 2020a), prediction of molecular structure (Grisoni et al., 2020)), the most natural sequence modelling task within AI is language modelling, on which there has been tremendous success in recent years. Language modelling and its fascinating recent developments will be discussed in more detail in a further section. Similarly to how generalist models can be pretrained with contrastive learning on large datasets, sequence modelling is widely used as pretraining on unlabeled data before the models are finetuned on tasks of interest. Self-supervised learning, because it can leverage the large amounts of unlabeled data found in the wild, is an essential component of most application of contemporary machine learning.

1.1.3 Reinforcement Learning and Deep RL

RL: an introduction

The third major machine learning paradigm is Reinforcement Learning, or RL (Sutton & Barto, 2018a). The RL setup is more complex than the ones of unsupervised and unsupervised learning, because it deals with the interaction between an agent and its environment, and with learning optimal sequential decision-making behavior from experience. No dataset, labeled or unlabeled, is given; the agent must play act, observe, and learn by trial-and-error.

RL is formalized as the question of an agent learning to behave optimally in a Markov Decision Process (MDP). An MDP is a discrete-time process formalized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, p, \gamma)$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space. The state space describes the set of states the agent can be in, and the action space symbolizes the actions the agent can take. $\mathcal{T}$ is called the probabilistic transition function, and if $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $\mathcal{T}(\cdot|s, a)$ is a probability distribution over $\mathcal{S}$; quantifying the probability of transitioning to a given state given that the agent was in state s and performed action a. R is the *reward function*. It is a scalar function defined over $\mathcal{S} \times \mathcal{A}$ and provides the optimization target for the RL problem. p is a probability distribution on $\mathcal{S}$ of initial states. The scalar $\gamma \in [0, 1]$

is the discount factor trading off immediate rewards for future rewards. A *policy* is a (potentially probabilistic) function from states to actions modelling the behavior of the agent. The RL problem is to find an optimal policy $\pi^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximises the expected discounted sum of rewards in a horizon T (potentially infinite). In other words, the RL problem is:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi, s_0}\Big[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\Big] \tag{1.3}$$

where the state-action sequence (called the trajectory) is sampled according to the agent's policy and the environment's transition function. This seemingly simple optimization target (a scalar function of the state and action) is actually a very general formulation that can be applied to setups as various as:

- **Continuous control**: there has been widespread success in controlling robots and simulated bodies through RL (Lillicrap et al., 2015). RL was applied as well to robotic control (Morimoto & Doya, 1998) and simulated continuous control benchmark are widely used in RL research (Todorov et al., 2012; Tassa et al., 2018). In these cases the reward quantifies some useful performance metric that we would like our model to exhibit;
- **Games**: Games lend themselves quite naturally to the MDP formalism and the reward function of game environments is straightforwardly victory or defeat in the game. Games have also had a long history in reinforcement learning, from antique checkers-playing programs (Samuel, 1959) to TD-Gammon (Tesauro, 1995) to the recent successes in the complex and subtle game of Go (Silver et al., 2016, 2017). Video games, especially the ones played at a very high competitive level (such as StarCraft II and Dota2) have also in recent years been the theater of impressive advances in RL;
- **Control of real-world physical systems**: The most prominent example of this is the training in simulation and real-life transfer of policies to control super-hot plasma inside a fusion reactor (Degrave et al., 2022);
- **Design and engineering**: RL can be applied to design problems that can be expressed as a sequential decision-making problem such as chip placement (Mirhoseini et al., 2020), or the low-level design of sorting algorithms (Mankowitz et al., 2023);
- **AI alignment with human preferences**: if one can derive a reward function that models human values, one can use it in an RL setup to align a pretrained language model's behavior with what is deemed acceptable by humans. This process is called reinforcement learning from human feedback (RLHF) and has successfully been used at large scale to constrain contemporary chatbots's responses to be helpful, polite and harmless.

There is an abundant zoology of RL methods that has been developed over the years, and a single book wouldn't be enough to cover the subject properly. In this short introduction we will only cover the subspace of RL relevant to this dissertation, and we refer to Sutton & Barto (2018b) for a more complete, but not exhaustive, treatment. RL methods can either be *model-based* or *model-free*. Model-based methods use a model of the environment (an estimation of the transition function $\mathcal{T}$) to plan ahead. Model-free methods are computationally simpler and are short-sighted. In this section we will focus on model-free methods which are the ones that have enjoyed the most success in

modern applications. There are 2 broad classes of model-free RL algorithms used in contemporary high-dimensional applications: first temporal difference learning (TD learning) based methods that rely on learning a value function (resp. action-value function, also called Q-function) associated with a policy, quantifying the expected discounted sum of rewards when following the policy when we are in a given state (resp. in a given state and performing a given action). These functions can be learned from experience and implicitly provide a policy. TD-learning methods will be covered in the next paragraph. The other broad class is the class of *policy optimization* methods and is only defined for a *parametrized policy* π_θ. It relies on an estimation of the gradient of the return (sum of rewards) with respect to the policy parameters. By ascending this gradient one can optimize the policy. A short introduction to policy gradient methods will be given after our discussion of TD-learning. *Deep Reinforcement Learning* (or Deep RL, DRL) refers to the use of neural networks for approximating the functions underlying the RL algorithm, for instance parametrized value functions, Q-functions, or policies. The use of neural nets as function approximators in RL allows generalization to novel states and handling very high-dimensional state spaces, such as image inputs. In the following overview of RL algorithms we will build up to Deep Q learning as a representative of Deep RL methods built on TD-learning, and to simple actor-critic algorithms as a representative of deep policy gradient methods.

Figure 1.3: A collection of screenshot of Atari games. The games, originally designed for players of this early console, became a testing ground for deep RL agents with discrete action spaces. Figure from Badia et al. (2020)

Q-Learning, DQN and followups

Temporal-difference learning based RL algorithms have their origin in Dynamic Programming (Bellman, 1957) over the value function. As we briefly mentioned above, the value function V^π for a policy π in a state s is the expected discounted cumulative sum of rewards when starting from s and sampling actions according to π: $V^\pi(s) = \mathbb{E}_\pi\big[\sum_t \gamma^t R(s_t, a_t) | s_0 = s\big]$. The optimal value function V^* is the value function for the optimal policy π^*. The value function for any π has an interesting consistency property that follows from its definition: if you take an action according to π, the environment transitions and you end up in state s' experiencing reward r, the sum of the reward you experienced and the discounted value for the new state should be equal to the value in the

previous state. Formally, for any given transition (s, a, r, s'), the value function obeys the following recursive identity, called a Bellman Equation:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi, s'\sim\mathcal{T}}\big[R(s,a) + \gamma V^{\pi}(s')\big] \tag{1.4}$$

The optimal value function obeys a Bellman optimality equation, reflecting the fact that acting optimally means selecting the action with the highest right-hand side:

$$V^{*}(s) = \max_{a} \mathbb{E}_{s'\sim\mathcal{T}}\big[R(s,a) + \gamma V^{*}(s')\big] \tag{1.5}$$

Q-learning is based on the Q-function, the expected returns knowing that we are in state s and have performed action a: $Q^{\pi}(s) = \mathbb{E}_{\pi}\big[\sum_t \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\big]$, and Q^{*} is the Q-function for the optimal policy. The Q-function also obeys a Bellman equation analogous to the one for the value function, with an addition of an expectation over actions. Finally, the Q-function also obeys a Bellman optimality equation:

$$Q^{*}(s,a) = R(s,a) + \gamma \max_{a'} \mathbb{E}_{s'\sim\mathcal{T}}\big[Q^{*}(s',a')\big] \tag{1.6}$$

Q-learning uses this identity for the optimal Q-function to propose updates for a suboptimal estimate. Namely, if Q is the current, non-optimal candidate, the difference between the left-hand side and the right-hand side of equation 1.6 applied to Q will give us the update term to the current Q-function. Learning Q is thus an optimization problem on collected transitions. The interesting feature of this learning process is that it uses the current estimate of Q to produce a new, better estimate, through interaction with the environment: in this context this is usually referred to as bootstrapping (in reference to the the famous Baron Munchausen pulling himself out of the river by his own bootstraps). Once the optimal Q-function has been found, we automatically get the corresponding policy: $\pi^{*}(s) = \arg\max_a Q^{*}(s,a)$. Note that this policy is deterministic. Q-learning proceeds by alternating action-value function learning and sample collecting. To collect further transitions, a policy based on the current estimate of the current Q-function is used.

The obvious choice would be to use the *greedy policy* corresponding to Q, that is, always select the action that maximises the current estimate Q. This is in reality a very poor choice, because of an issue specific to reinforcement learning called the *exploration-exploitation tradeoff*. This problem, fundamental to learning from experience, arises because RL methods both need to both learn how to act and act to collect their own data for learning at the same time. When high reward has been observed in the past for some action, should we keep repeating this action to maximize rewards under our current knowledge, or take the risk of exploring other options with potentially less reward to gain additional knowledge, leading to more reward in the long run? The RL literature abounds with discussions of the explore-exploit tradeoff and of exploration methods to mitigate them. The simplest exploration method for Q-learning is ϵ-greedy: at each step, selecting a random action with probability ϵ. We will discuss exploration in RL in the later section dedicated to intrinsically-motivated reinforcement learning, in Section 1.2.3. We have focused here on Q-learning since it was the first value learning method to be combined with deep learning (Mnih et al., 2015), but many other variants exist, based either on value functions or on action-value functions, such as TD(λ) or Sarsa (Sutton & Barto, 2018b).

Deep Q-learning parametrizes the Q-function with a neural network to be able to learn flexible policies for complex control problems in high-dimensional spaces. The idea of combining Q-learning

and neural nets is quite old, since the algorithm naturally lends itself to stochastic gradient descent on the Bellman error. However, learning is especially unstable, suffering from exploding Q-function estimates and catastrophic forgetting, since, as always in RL, the distribution of data on which the model trains is shifting over time as the agent learns and experiences different regions of the MDP. The first working version of deep Q-learning, called the Deep Q-Network (DQN) was a breakthrough (Mnih et al., 2015), and was demonstrated on Atari games where the agent reached superhuman performance. Among the innovations introduced in DQN were the use of a *replay buffer*, a dataset of previously encountered transitions that the algorithm samples from to estimate the Bellman error on minibatches that are less correlated than online samples. Another stabilizing trick was using a frozen target model for the right-hand side of Equation 1.6 when computing the TD error. After the success of DQN many other improvements were proposed (Schaul et al., 2015c; Van Hasselt et al., 2016; Wang et al., 2016; Bellemare et al., 2017; Fortunato et al., 2017), which provided even more performance when combined together (Hessel et al., 2018).

Policy gradients

The other influential thread in recent model-free RL research are *policy optimization* methods. Instead of learning an action-value function and of acting according to it, policy optimization (sometimes called policy-gradient) methods directly try to optimize a policy function. In a sense these methods are more direct: they directly optimize the RL objective. They rely on the fact that if one uses a parametrized policy π_θ, one can directly estimate the gradient of the returns with respect to θ, which lends itself well to gradient ascent. This is done in two stages: first by deriving an analytical expression of the expectation of the gradient of the returns $\operatorname{grad}_\theta J_\theta$ then by finding a way to sample from it. There are many unbiased estimators, differing widely in their variance. The seminal paper in that regard was the REINFORCE paper that proposed the following expression (this is given for the episodic case):

$$\operatorname*{grad}_\theta J_\theta = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T} \operatorname{grad} \log \pi_\theta(a_t|s_t)\, G_\pi\Big] \tag{1.7}$$

with:

$$G_\pi = \sum_{t=0}^{T} R(s_t, a_t) \tag{1.8}$$

The sum of rewards during the whole episode. Subsequent work has found alternative expressions for G_t with lower variance, such as the rewards-to-go $\sum_{t'=t}^{T} R(s_{t'}, a_{t'})$, the on-policy action-value function $Q^\pi(s_t, a_t)$, any version of these with the value function subtracted (the value function in this case is called a *baseline*), or, finally the advantage function for the policy π: $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$. Policy gradient methods that use an estimate of the value, action-value or advantage function are called actor-critic methods (Mnih et al., 2016) and have enjoyed widespread success in the the Deep RL literature and in applications. Further developments in policy gradient methods was the invention of Trust Region Policy Optimization (Schulman et al., 2015), a method for performing large parameter updates on the policy while keeping the newly updated policy sufficiently close to the old policy in policy space (in a trust region according to KL-divergence). Proximal Policy Optimization (PPO) (Schulman et al., 2017), a very widely used deep RL algorithm is an approximation of TRPO, where the optimization problem to solve to keep the updated policy close to the old policy is made significantly computationally lighter.

There are some advantages of policy gradient methods over action-value methods (Sutton & Barto, 2018b). For one, the policy might actually be a simpler object to approximate than the Q-function. Another argument is that it is naturally easier to represent stochastic policies directly than to turn an action-value function into a stochastic policy. And a last advantage is that policies might benefit from prior knowledge: if we have some expert trajectories in a particular environment, supervised learning of actions conditioned on previous states and actions (like in supervised/self-supervised sequence modelling discussed in the previous section) can be a very effective way to initialize a policy with very good priors that can then be plugged into an environment and further trained with reinforcement learning. On the other hand, policy gradient methods, because they rely at their core on an estimate of the gradients with respect to θ *under the current policy* are always on-policy (while value methods are off-policy, which means they can use old transitions to fit the Bellman objective). The on-policy nature of policy gradient methods prevents us from re-using old samples. For a single-agent task, this might be all right; but for learning multi-task policies one might need to continually train on old data to remember tasks experienced at the beginning of training (Rolnick et al., 2019).

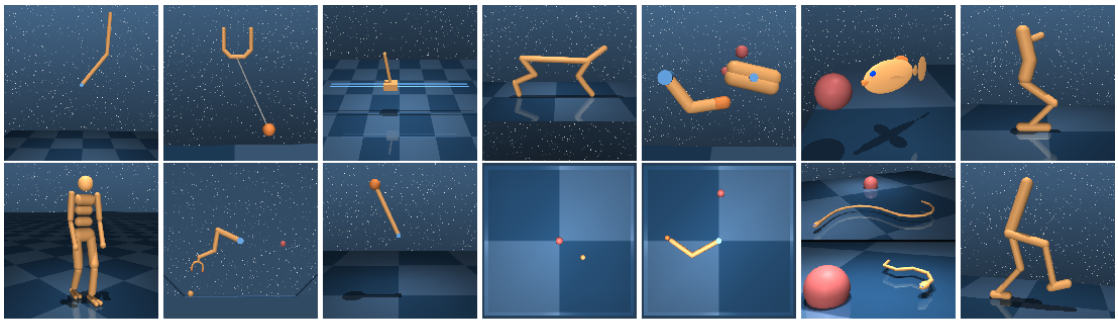


Figure 1.4: The DeepMind control suite started became a standard benchmark in continuous reinforcement learning (image from Tassa et al. (2018))

Once major Deep RL algorithms started working, innovation in the neural architectures used in RL agents could flourish. An important class of architectural innovations in deep RL was the use of sequence models or recurrent models to implement memory in agents. The Markov assumption of MDPs is most often not valid in realistic environments: there might be hidden information that determines how the states transition. Partially-Observable Markov Decision Processes (POMDPs: Kaelbling et al. (1998)) are a formalism for describing such cases: the model includes a state s with all information and a partial observation o that the agent has access to through an observation function $\mathcal{O}$ such that $o \sim \mathcal{O}(s)$. One way of dealing with POMDPs is to use a policy/Q-function that incorporates memory of the past states (Hausknecht & Stone, 2015). More recently, following the successes of the Transformer (Vaswani et al., 2017) architectures at sequence modelling, RL agents with Transformer architectures have been implemented (Parisotto et al., 2020). Other important and general improvements in RL have come from auxiliary tasks for training helpful representations within the networks; representations that might help train the policy or value functions (Jaderberg et al., 2016).

1.1.4 Self play, multi agent systems

As we have seen, the mid-2010s in AI were the moment when deep RL methods really started to shine, and some of the most impressive subsequent breakthroughs in AI until the early 2020s were in game-playing AI (Silver et al., 2016, 2017; Vinyals et al., 2019; Berner et al., 2019; Bakhtin et al., 2022). This is a nice echo of the pioneering work in AI in the 1960s, where the first breakthroughs were also in game-playing. However, what's particularly interesting to us about game-playing is the use of a training technique called self-play (Silver et al., 2017). Imagine that you had to train a superhuman chess-playing agent with RL. How do you define the environment? The environment has to include the opponent, because from the agent's point of view the opponent's move is the environment's transition function. Now suppose you start off by playing against a weak opponent: by the end of training the agent has mastered this environment and can consistently beat the weak opponent. But after that the agent's performance will plateau: there is no further reward to be obtained in becoming better: the opponent's level is its skill horizon. A natural fix is to have the agent play and try to beat a really strong opponent. However, in this case the untrained agent is simply too weak to experience any rewards because it will consistently get beaten by its opponent: and in RL no reward means no learning. Our agent won't even learn how to play. Another option would be to have it play first a weak opponent, and then present it gradually more competent ones (in a similar fashion to how humans might learn to play a game by trying different difficulty levels in a sequence), a technique called *curriculum learning* (Elman, 1993). There are however 2 issues with this approach: first, you don't know at what rate your agent is learning, to determine the schedule of opponents. And secondly, you need to provide a library of opponents spanning a whole range of difficulty levels to your agent, which implies you already have solved part of the challenge of building competent agents on this game. Self-play deals away with these difficulties. As the agent plays against itself, both sides are learning. This elegantly solves the issues of providing opponents and scheduling a curriculum, and it is a form of *automated curriculum learning* (or ACL: (Portelas et al., 2021), see also discussion in Section 1.2.3).

An extension to self-play is systems that maintain a pool of agents initialized from the same baseline and are evolved together by playing against each other with multi-agent reinforcement learning (MARL: Littman (1994)). We'll describe shortly one such approach that we find very interesting: DeepMind's AlphaStar algorithm. AlphaStar is an approach to try to play the competitive game StarCraft II (SC2), where the goal is to control units such as workers to gather resources and build factories, and armies whose goal is to destroy the enemy's base[1]. This real-time strategy game is considered to be one of the most challenging e-sports as the players have to, at the same time, micromanage the movement of their units, scout the map to reveal information about the enemy's position and current strategy, handle the tactics for potentially several battlefields at once, control the map by placing buildings, plan and commit to a general strategy to counter the opponent by building the right tech, and apply knowledge of what strategies players are likely to play, which depends on updates by the game developers and strategy and counter-strategy finds by the community of SC2 gamers (this strategy zeitgeist evolves slowly and is called the *meta*). This (alongside the high-dimensional observation and action space and the high level of partial observability) makes StarCraft II an impressive challenge for RL. AlphaStar's approach consisted of 2 steps: the first

[1]The author of these lines personally enjoyed following the StarCraft competitive scene, but also finds the surveyed work scientifically significant because it addresses the issue of maintaining a diversity of behaviors, which is very relevant for this thesis.

was to imitate the trajectories of human expert players. This was required to get the agents to even finish the game: random play would never lead to any signal, so vast is the action space. After this initial pretraining the AI players reached a very reasonable prior on how to play the game, leading to moderate performance. Getting the agents to play at a competitive (grandmaster) level however was much harder. The authors defined a pool of agents and designed a league system wherein agents from different leagues had different objectives: some had to beat the maximum number of other agents from all leagues while others should focus on beating agents from a particular league, or the top performing agents. This population of agents with a diversity of objectives ensured that counter-strategies were effectively found to beat entrenched strategies and that enough diversity was kept as training proceeded. This also meant that human pro players could not learn to exploit the learned strategy as easily as they could have done if there only was one agent being trained. This multi-agent aspect implemented a version of co-evolution (Arulkumaran et al., 2019), a notion that we will discuss in more detail in our discussion of evolution as an open-ended process in Section 1.2.1. Another prominent example of co-evolution of strategies and counter-strategies in a MARL setup is given by the work of Baker et al. (2019), wherein long training times lead to the emergence of an increasingly complex set of behaviors.

1.1.5 Language modelling, Transformers, and scale

Language Models (LMs)

After this foray in RL and the advances it has it has permitted, we will end our account of the recent advances in AI with the last important development in deep learning, beginning towards the end of the 2010s: the unstoppable rise of large language models (LLMs), and large generative models in general (sometimes also called foundation models (Bommasani et al., 2021)). If you recall our general introduction of self-supervised learning, we defined sequence modelling tasks: predicting one or a set of elements of the sequence based on other elements. This paradigm, applied to language, has had a revolutionary effect on machine learning, because, as mentioned before, this generic way of training captures useful representations that can be used in a wide variety of downstream tasks. Language modelling is usually separated in two categories:

- **Masked language modelling**: In this task, a text is broken down in its constituent elements (*tokens*, which can be characters or more often sub-word elements which are strings of characters that co-occur often in a text). Some tokens (around 20%) are masked with a special mask token, and the task of the language model is to predict the masked tokens conditioned on the tokens it has access to. The most well-known model in this category is BERT (Devlin et al., 2018). These models are usually used for capturing representations for downstream tasks;
- **Autoregressive language modelling**: In this task the n first tokens are given and the model's task is to predict the next one. The advantage of this form of language modelling is that one can sample from the distribution of next tokens, add this token to the list, and repeat. The most well-known examples of this technique are the Generative Pretrained Transformers (GPT: Radford & Narasimhan (2018); Radford et al. (2019); Brown et al. (2020)) series. This form of language modelling is also called causal language modelling, because the causes (early tokens) lead to the consequences (subsequent tokens).

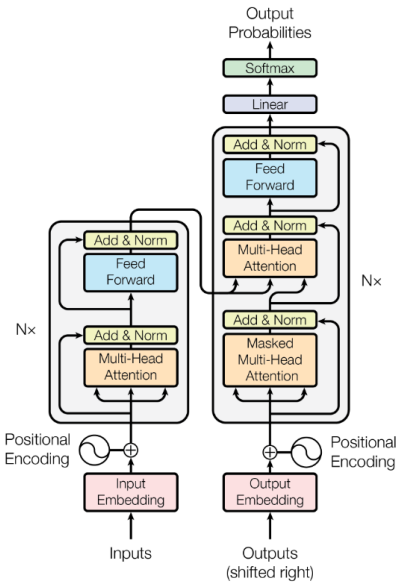


Figure 1.5: Classical illustration of the Transformer architecture from Vaswani et al. (2017). The original architecture was an encoder-decoder whu=ich you can both see here. The GPT series only has a (causal) decoder.

This research thread started when researchers realized that training a causal language model to predict the next token in a sequence yielded clear representations of the text and reasonable generative models. One of the first milestones in this thread was the discovery of the *unsupervised sentiment neuron* in a causal LM trained to predict text on a sentiment analysis dataset (Radford et al., 2017). This neuron reliably learned to recognize sentiment from the text while not explicitly trained to do so: the only training objective was to predict the next token. However, the limitations of LSTMs blocked scaling the approach. Since recurrent neural networks need the hidden state of the previous step to compute predictions for the next step, training necessarily unfolds sequentially, which is extremely slow. This is a considerable bottleneck in training these sorts of models on large amounts of data.

1.1.6 Transformers

A few years prior, it had been noticed that the performance of LSTMs in seq2seq tasks was greatly improved by adding attention connections that skipped the temporal computation inherent in recurrent models (Bahdanau et al., 2014). In a parallel thread of research, work started to appear that concentrated on set prediction: a task similar to sequence modelling but without any temporal structure attached to it (Santoro et al., 2017b; Battaglia et al., 2018a): set predictions should be either invariant or equivariant to permutation (respectively not change when a permutation is applied, or change in the same way). This culminated in the invention of the Transformer (Vaswani et al., 2017): a model that revolutionized sequence modelling and is the powerhouse behind all the industrial applications of generative models we see today.

Transformers are set models that take as input a set of embeddings $(x_i)_{i\in[1..n]}, x_i \in \mathbb{R}^d$ and return another set of embeddings $(y_i)_{i\in[1..n]}, y_i \in \mathbb{R}^{d'}$. Transformers are invariant to permutation: for any permutation σ, $\mathrm{Transformer}(x_{\sigma(i)}) = y_{\sigma i}$. Transformer encoders and decoders are made of stacks of layers. Each layer is made of a *multi-head self-attention layer*, a skip-connection, LayerNorm (Ba

et al., 2016), feedforward (multi-layer-perceptron: MLP), skip connection and another LayerNorm. The multihead self-attention layer was the key innovation. A single-head self-attention layer computes, for each x_i a set of keys k_i, queries q_i and values v_i, each one computed through a different linear projection. For each element at position i, the attention between it and the element at position j, a_{ij} is computed by taking the j-wise softmax of the scalar product between q_i and k_j. The element at position i is updated by replacing it with the sum of all values v_j weighted by the attentions a_{ij}. In other words, if we denote here with $x_i^{(t)}$ the vector at position i before the self-attention layer, and y_i the value after the self attention layer, Q the query projection, K the key projection and V the value projection, we have:

$$y_i = \sum_j \mathrm{softmax}_j\left(x_i^\top Q^\top K x_j\right) V x_j \tag{1.9}$$

The multi-head self-attention layer simply parallelizes this matrix multiplication operation h times, to allow the key, query and value vectors to encode different operations. The vectors x_i are simply split into h segments, each one undergoes multi-head attention with a set of matrices unique to this head, and the resulting vectors are concatenated back together.

The intuition behind the Transformer is that there needs to be 2 separate kinds of computation on a set of vectors: independent, parameter-shared functions on the tokens (performed by the MLPs), and computation that allows mixing between information originating from different tokens. The self-attention matrix (a_{ij}) quantifies how much the information from element j is important to the next value of element i.

The reader will have noticed the Transformer is an equivariant network: permuting all elements except element i should have no effect on y_i. However this is not a realistic model for language: *the cat ate the dog* and *the dog ate the cat* are two very different sentences and this should be reflected in the model. To incorporate this sequential information, Transformers make use of a so-called *positional encoding* (PE): a set of n vectors uniquely encoding their position. The original paper proposed a continuous generalization of the binary encoding of a number, in the form of a vector whose elements are sinusoid functions of the position with increasing frequency. Later works proposed to learn the positional encodings directly, and other types of PEs with various properties have been introduced (for instance the rotary PEs of Su et al. (2021)).

The impact of Transformers has been decisive in deep learning research, and in less than five years the original paper has become one of the most cited papers of all time. The reason for this widespread success is twofold: Transformers, through the attention mechanism, allows flow of information between distant positions, enabling long-term dependency. And second, and very important point, is that this architecture can be fully parallelized on GPUs. RNNs needed irreducible sequential computation to be trained: Transformers on the other hand can be scaled up to immense number of parameters and data, which ushered in the scaling era of machine learning.

1.1.7 Scaling laws

Following the discovery of the unsupervised sentiment neuron and the invention of the Transformer, researchers started combining both inventions together to make Transformer-based language models. This was the start of the Generative Pretrained Transformer (GPT) series that gained increasing capabilities and impact. Using Transformer architectures allowed the researchers to train on

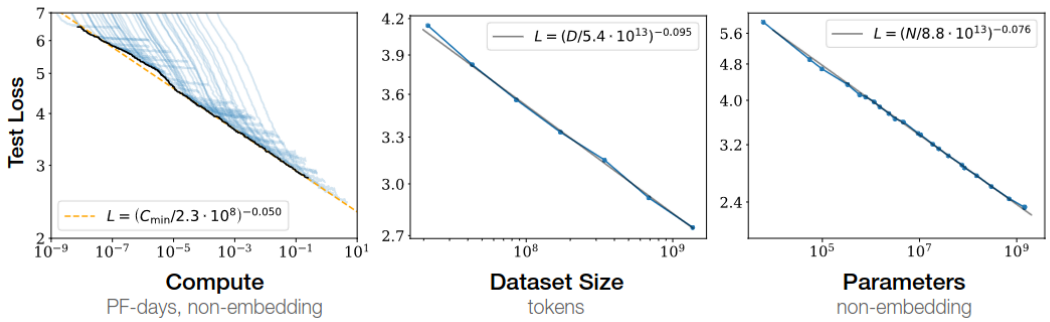


Figure 1.6: Kaplan scaling laws for LLMs with respect to number of parameters, data, and compute. The laws are valid for one of the variables when not bottlenecked by the other two. These laws led to the training of very large models (> 100B parameters) for few tokens (~ 300B). Figure from Kaplan et al. (2020).

more text with larger models, and over time this led to fundamental discoveries. The main lesson learned was that with increasing scale in parameters and data the model started to acquire emergent capabilities not displayed at lower scales (Brown et al., 2020). Scaling up LMs brought them multitask abilities, as well as *in-context learning*, which is the ability to learn new tasks on-the-fly without any gradient updates. This is done by feeding the LM *few-shot* examples of tasks in its *prompt*, the text used to condition further data generation. Language models are one of the most compelling examples of truly multitask models (in language domains), due to the general knowledge needed to faithfully model the conditional distribution on next tokens over internet-scale textual datasets. This is significant and we will discuss it in the experimental section as well as in the discussion section of this manuscript.

The emergence of new capabilities in large language models based on Transformers seemed mysterious and unexplainable. A line of research started looking at mathematical relationships between various hyperparameters of training and the final performance. The seminal work (Kaplan et al., 2020) found precise, smooth power laws between final performance (measured as the loss on a test dataset) and a model's number of parameters N, the dataset size in tokens D, and amount of compute C. The resulting *Kaplan scaling laws* suggested that models should be made larger and their training stopped sort of convergence, because larger models are much more sample-efficient. These power laws are reproduced in Figure 1.7, and started informing the design of the largest models (Rae et al., 2021; Zhang et al., 2022; Chowdhery et al., 2022; OpenAI, 2023). An important chapter in the scaling saga was the publication of the *Chinchilla scaling laws* (Hoffmann et al., 2022). In this paper, the authors reformulated the question of determining N and D, given a fixed compute budget. They found another relation (also due to varying training details, such as the learning rate schedule, with scale) that suggested compute-optimal models should be much smaller than originally believed, and while still not trained to convergence, they should be trained for substantially larger number of tokens. The authors then used this relationship to train a model (Chinchilla, in the reference to the small, fast, and cute animal) with comparatively few parameters but trained on a massively larger number of tokens (70 billion parameters compared to 130-500 billion for existing models at the time; 1.4 trillion tokens compared to 300 billion for models at the time). This paper was instrumental in determining the data for further smaller models such as the Llama series (Touvron et al., 2023a,b) which had a wide diffusion and due to their small size and availability sparked much subsequent academic and

open-source work (Taori et al., 2023; Chiang et al., 2023; Xu et al., 2023). Chinchilla scaling laws, if they stay valid outside of the regime where they have been derived, might suggest that in the near future data, not compute, could be the limit for performance of LLMs[2].

Reinforcement Learning and Language Models

While the wider ML community had recognized the abilities and promise of LLMs for a few years, these models stayed below the radar for the general public as even powerful LLMs, despite their abilities, are pretty hard to use. One of the primary ways to interact with a linguistic agent is to give it instructions (a goal) and expect the agent to execute these instructions or answer the question. However, raw LLMs are not trained for this task, they are only modelling the probability of the next sentences. A seminal advance in building instructable models was the InstructGPT paper (Ouyang et al., 2022). The authors framed the finetuning of a language model to follow instructions as an reinforcement learning problem. The key insight here was to use as a reward function a reward model trained to recognize whether a given text completion was an answer to the request or not. This reward function was trained on a collection of instructions, each instruction being supplemented with a set of completions. Human annotators were asked to produce rankings of these completions according to how well the completions answered the instruction, and the reward function was trained to predict these rankings. More generally, the paradigm of training a reward model from human preference data and using it to finetune a language model with reinforcement learning is called reinforcement learning from human feedback (RLHF). This kind of finetuning has become very widespread in recent years and has allowed the creation of chatbots specifically made for conversation-like interaction (such as ChatGPT, Claude, etc) that have achieved extremely widespread success with the general public.

RLHF has also been massively used to make these models more save and generally *aligned* with human values. AI alignment is a subfield of AI that is interested in methods for creating AI systems whose goals are compatible with ours. Historically, the field often concentrated on speculative and theoretical arguments (Bostrom, 2014b), but more recently practical alignment techniques have started to emerge. One prominent example is the alignment from IRL proposition of **?**. In this view, one could extract a utility function quantifying human values from human behavior through a set of techniques called Inverse Reinforcement Learning (IRL: Ng & Russell (2000)). This reward function can then be used to align the model through RL. This is very close to RLHF methods, and the seminal paper on RLHF (Christiano et al., 2017a) explicitly motivated RLHF through the lens of alignment. Alignment-focused RLHF finetuning has yielded models that are difficult to use to write spam, harmful or hateful content, or to get help with illegal information. These models often can be "jailbreaked", meaning they can be tricked into producing content discouraged by RLHF training. However, continuous improvement based on user feedback (at least on the larger, more popular models) means the jailbreaks are usually fixed and new ones become harder to find by hand.

What the success of RLHF as a training paradigm shows more generally is that one can use reinforcement learning at scale to finetune pretrained language models. RL on large, transformer-based models is notoriously hard and unstable (Parisotto et al., 2020). However, massive self-supervised training gives these models excellent priors on linguistic tasks. Using policy gradient methods with a policy initialized with these general multitask agents, on text-based MDPs, provides

[2] as was argued recently in this blog post extrapolating the data requirements and available number of tokens following the scaling laws: https://www.alignmentforum.org/posts/6Fpvch8RR29qLEWNH/chinchilla-s-wild-implications

immense benefits compared with training from scratch. We are reminded of the training regime of the AlphaStar (Vinyals et al., 2019) agents that were first pretrained on expert trajectories and then finetuned with multi-agent RL. RL on language models (RLLM) is the transposition of this training paradigm in the language domain (with no multi-agent dynamics). Some recent examples of RLLM with general, task-based reward functions are to be found in coding domains (Le et al., 2022) where execution can provide a basis for task completion, as well as text-based games such as in (Carta et al., 2023).

1.1.8 An important missing piece: creative curiosity and problem-finding

AI research has, from its origins in the middle of the 20th century, made immense progress. As forseen by some of the early AI researchers, the main ingredient seems to have been the increasing availability of computation coupled with the invention of very general methods that could leverage this computation. Machine learning is the paramount example: researchers have crafted a way for machines to learn complex knowledge from data directly, instead of relying on the imperfect communication of domain knowledge we could provide. For instance, instead of relying on principles of linguistics derived from the study of language, NLP has been solved by massively training models on large amounts of text mined from the internet. Through deep reinforcement learning and self-play we can build agents that can learn to play arbitrary games, with or without perfect information, in cooperative of adversarial setups, at a superhuman level (Silver et al., 2017). Through large-scale imitation learning, we can train massively multitask models possessing knowledge in many different domains and capable to efficiently execute arbitrary human-given queries (Schulman et al., 2022).

What is missing from AI today is the ability for open-ended discovery. Current training paradigms are not adapted for autonomous exploration. Unsupervised, self-supervised and supervised training all assume there is some underlying data distribution with representative examples of the task to be performed. The best a model can do in these paradigms is perfect interpolation: fitting the known data. One can ask an LLM to write an explanation of deep reinforcement learning in iambic pentameter, but if you let one of these models generate freeform text without any particular predefined task, there is little chance that they will invent new challenging tasks for themselves and train to be successful at them, the way any bored child might do. *An LLM left to itself does not play*. Creativity in LLMs comes from the user's prompt, not from the model itself. Good completions, and trial-and-error, come from the user's curation, not from the model itself. An additional obstacle to LLMs' open-ended discovery abilities is the lack of ground truth. When performing trial-and-error to invent something new, humans generally have feedback from an environment, but LLMs usually interact only with themselves when training or in inference. RLHF training is a preliminary study in providing LLMs with an interaction loop, and there are some works beginning to study tool use (Schick et al., 2023) in-context feedback (Shinn et al., 2023) but more progress might be needed. Something is missing from large-scale supervised models.

On the other hand, deep reinforcement learning agents are made to make novel discoveries by trial-and-error. Instantiated in a game, they display the same kind of creative use of game exploits to gather as much points as possible as the most ingenious of speedrunners[3]. The discovery potential of deep RL agents was especially visible in the landmark Go match of AlphaGo versus Lee Sedol, the

[3] Speedrunners are players trying to finish video games (possibly with arbitrary constraints.) as fast as possible by taking advantage of game engine bugs

Go world champion, in 2016. Early in the second game of the series, AlphaGo made a very surprising move that came to be called Move 37. The position the machine played was unconventional and widely regarded by commentators as quite bad; however later in the game this move proved to be instrumental in AlphaGo's victory. Lee Sedol himself said the machine exhibited, in this case and in others, a form of creative playing. By training against itself AlphaGo had explored a subspace of the space of strategies that was unknown to the community of human Go players, playing against one another for thousand of years. However, deep RL agents are limited to one particular domain. Go-playing agents cannot play StarCraft, hold conversation or write a haiku: they do not have the generality displayed by LLMs. And more importantly, however good at playing these agents are, they are not inventing their own games to play as any five-year-old will do effortlessly.

In other words, a certain form of creativity or curiosity is missing from today's agents and models. They have flexibility and creativity in solving problems, but they do not invent problems for themselves. They do not challenge themselves to learn more. Beyond games and NLP tasks, the desire and ability to create one's own challenges is what underpins science and mathematics, as scientists ask themselves very difficult questions and the community slowly works together to answer the questions or prove the theorems, sometimes over hundred of years. The ability to challenge oneself and invent new forms is also one of the main components of art (especially avant-garde art): artists use their understanding of current trends and themes to express emotion and affect using new forms and new techniques. The ability to creatively build one's own problems is also at the core of a part of business creation, where good entrepreneurs come up with novel, disruptive and realistic ideas for a company and thus expand the frontier of what material goods or services are available to individuals or companies (Thiel & Masters, 2014).

1.1.9 Intrinsically-motivated open-ended learning

The aim of this thesis is to provide some elements of insight on how to build the drive for discovery in AI agents. To do so, we will ground our theoretical and experimental contributions in the field of developmental AI. *Developmental psychology* (See also Section 1.2.3) is the study of the learning processes and developmental stages of children. *Developmental robotics* (Baldassarre & Mirolli, 2012) is a field, created in the mid-2000s, aimed at constructing general cognitive models of children's learning and validating these models by implementing them in embodied robots and observing their learning abilities. *Developmental AI* shares the goals of developmental robotics but uses general-purpose machine learning as the computational tools to do so. The relationship between developmental AI and traditional AI is interesting. AI is guided by progresses in problem-solving. Progress is usually guided by setting hard benchmarks and trying to collectively solve them (remember the role of the Imagenet challenge (Russakovsky et al., 2015) in driving progress on computer vision tasks, or the target of mastering the game of chess and Go). However this focus on problem-solving is philosophically at odds with studying the problem-generating abilities of artificial agents. Setting a benchmark means the problem has already been invented, it sidesteps the creative curiosity problem. Developmental AI's goal is slightly underdefined: study, at a high level, the *developmental processes leading children to acquire an open-ended repertoire of skills*, and compare the outcomes of several such processes on the effectiveness of exploration, the diversity of learned skills and the amount of mastery the agents display in each discovered domain. Recurrent questions in developmental AI arise in the evaluation of such agents, and this issue will be discussed at several points in the experimental sections of manuscript, as well as in the discussion at the end. The well-defined aspect of evaluation is what we lose when going from problem-solving AI to developmental AI.

The overarching field studying intrinsic motivation, curiosity, creativity, play, and the invention of problems in humans and machines is sometimes referred to as intrinsically-motivated open-ended learning (IMOL). The history of the study of intrinsic motivation and the broad existing categories of intrinsic motivation will be outlined in more detail in Sections 1.2.3 and 1.2.3. Before diving into intrinsic motivation however, we want to spend some time discussing *open-endedness*. The study of open-ended processes is the study of algorithms producing interesting outcomes forever. We will begin with this perspective, first by discussing one of the open-ended processes that has existed for the most time, natural evolution through natural selection. We will then extend this discussion towards cultural evolution, which shares many commonalities with natural evolution. Noting that cultural variation must be driven by some process f cultural invention, we will be ready to discuss intrinsically-motivated agents, reviewing existing work in this domain and covering in particular a learning setting where agents are built so as to be motivated to invent their own problems. This setting is called *autotelic learning* (Colas et al., 2022c). We will conclude the section by motivating how autotelic learning can help make progress on AI systems inventing their own problems, thereby helping build AI systems that can be motivated to generate their own challenges and train themselves to solve them.

1.2 Open-endedness and intrinsic motivation

Let us begin our short introduction to intrinsically motivated, open-ended learning. The objective of this section will be first to present some organizing principles of natural evolution as a salient example of open-ended process. We will then shortly discuss the links to cultural evolution to motivate the importance of the concept of intrinsic motivation, and will then briefly review the theories and computational models of intrinsic motivation, before focusing on one such model which is the framework within which this thesis is set: *autotelic learning* (Colas et al., 2022c). One of the arguments we will be making is that intrinsic motivation in humans is the driving force behind cultural innovation, through creative invention. The study of intrinsic motivation and play is most present in developmental psychology, when studying the impressive exploration and learning abilities of children; we will discuss shortly some prominent theories from this area of research. We will finish by outlining how this setup provides elements for making progress on truly open-ended AI systems.

1.2.1 Evolution

Endless forms most beautiful

Life provides the most striking example of boundless, perpetually evolving complexity in our universe. The origins of life itself are mysterious, but it might have emerged around 4 billion years ago and developed in the hydrothermal vents providing energy and minerals in the hostile environment of the primordial Earth. Whatever its origins, life now covers all the environments, from the oceans to the land and soil and the air, and even the most hostile places have their share of expertly adapted extremophile lifeforms. What's more, the complexity of lifeforms seems to be ever increasing: multicellular life first emerged around 3 billion years ago (and repeatedly emerged in unrelated species at least 24 additional times), allowing for cell differentiation and large organisms with intricate morphologies and behavior, societies of organisms developed, then complex ecologies where multiple species interact and rely on each other in different ways, cooperating, competing, or

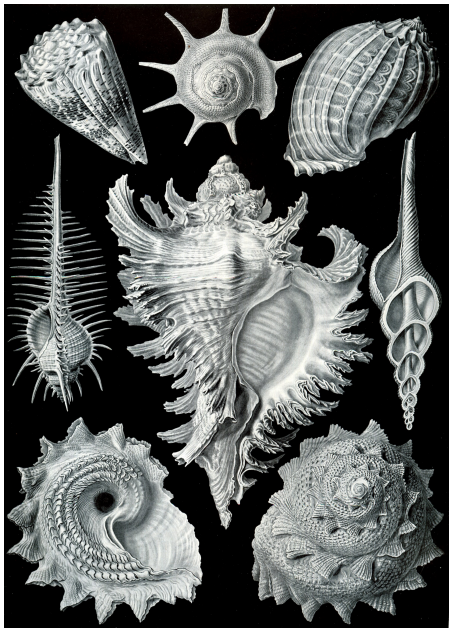


Figure 1.7: Natural evolution is the most eloquent example of an open-ended process. Illustration from Ernst Haeckel's classical *Kunstformen der Natur*.

taking advantage of each other.

Complexity of life forms is a hard-to-define concept: the number of actions available to the organism, its behavioral complexity, the number of genes it possesses, or its number of cell types are interesting measures that have been considered by the community of biologists (Szathmáry & Smith, 1995), as is the degree of agentivity of an organism (understood as the level of flexibility of the organism's behavior as based on its observations: (Tomasello, 2022)). Whatever the complexity measure however, as time passes and evolution advances, forms of life with ever increasing complexity seem to emerge. We will try to give a sense of what some of the mechanisms for the endless evolution of new species.

Variation, conservation, and selection

The simplest model of how natural evolution works is the one that has emerged at the time of the Modern Synthesis (Dawkins, 1976), when Charles Darwin's theory of evolution by natural selection was integrated with Mendel's law of inheritance. Darwin established that individuals underwent random variation that affected their ability to survive and reproduce, known as their reproductive fitness. Individuals would then, when reproducing, pass on traits genetically to their offspring, and in this way adaptive traits would spread and maladaptive traits would die out. However Darwin did not know about what was the support of inherited traits and believed transmitted traits would be the average of traits of parents. Mendel was the one to establish, in the middle of the 19th century, the existence of genes whose discrete alleles were mixed in reproduction. The Modern synthesis provides the standard model of evolutionary biology, and emphasises that the gene is the relevant level of selection: mutations and adaptations at different scales should be studied primarily through the way they affect the propagation of genes (Dawkins, 1976). The major components of biological evolution are *variation*, *transmission*, and *selection*. Random variation between alleles affect differential fitness and the proportion of alleles in the next generation. Less fit alleles get eliminated from the next

generation because the organism carrying them fails to survive or reproduce (selection). Organisms that do manage to reproduce pass on their genes (transmission), and either through random mutations either through sexual recombination the genes (or chromosome) passed to the next generation will be slightly different (variation), producing slightly different behaviors. Natural evolution is very complex, and there are exceptions to every rule, but that is a good rough picture of the evolutionary process. The impact of genetic changes has itself become very organized, as multicellular individuals go through morphogenesis to grow from a single cell to an intricate organism: impact of one of the genes that helps organize morphogenesis can have disproportionate effect on form (Carroll, 2006a). The selection pressures are organized at multiple scales, as with time genes, cells, or organisms started cooperating and reproducing as a whole (Szathmáry & Smith, 1995).

Punctuated equilibria and feedback loops

If we follow these principles we should mostly expect to see intermediate life forms between all present species in the present, and looking back in time we should expect to see intermediate stages between all species that existed in the past. However, today all multicellular life forms are separated in distinct species with very similar individuals, and the fossil record only contains sudden jumps from one species to the next, not all the intermediate life forms. The puzzle of the existence of species is sometimes known as Darwin's paradox. Gould & Eldredge (1977), after examining the discontinuous fossil record, proposed for the first time the notion of punctuated equilibria to describe the evolution of species. In the punctuated equilibrium theory, short bursts of speciation events are separated by long periods of stability where a species' form would be conserved for a long time. This contrasts with the so-called gradualist model, supported by Darwin himself, that posits that variation in animal morphologies happen by slow, gradual accumulation of mutations that, over millions of years, give rise to a change in form.

The reasons for stasis are comparatively less complex than the ones for rapid change. One of the foremost reason for long periods of stability in form (that might still involve invisible genetic changes within a species) in evolution is that of stabilizing pressures: pressures in the environment that makes variations have low fitness. An illustration would be color changes in white-coated animals living in arctic environments: both in prey and predator this would make individuals conspicuously visible, which would decrease the chances of survival. A related pressure is that of *koinophilia* (Koeslag, 1990, 1995) in sexual selection: individual preferring mates that have average appearance (probably because it is indicative of a low rate of deleterious mutations). Another effect is simply group size: larger groups evolve more slowly since genetic drift is slower with a large amount of genes in the pool. In contrast, causes for rapid variation can be multiple: migration of a species to a new place, rapid ecological or physical changes, or a small subpopulation being isolated for a while. In the next section we will make a rapid overview of a specific category of causes of rapid variation in species: ecological feedback loops.

When a feedback loop is created in evolution, the selection pressures evolve as the organism itself adapts, and both processes have the potential to accelerate each other. Feedback between organism and environment is often discussed in studies of *niche construction* (Odling-Smee et al., 2003). This refers to the process of an organism altering its ecological niche either as a central part of its behavior or as a side-effect, and this alteration of its ecology has repercussion of the organism itself by altering the selection pressures to which it is subject. The organism's existence and behavior can modify its environment long-term (a form of non-genetic heredity). Niche construction is a very powerful mechanism for evolutionary change. Beavers are an obvious example: they cut down trees to build

dams which then greatly modify the ecology of the river by it into a lake. Other well-known examples include pine trees shedding their needles to increase the chance of wildfires, thereby eliminating the competition of non-resistant plants; lemon ants cultivating tree groves of hundred of trees that are well-suited for their immense colonies; or earthworms changing the ph of soil so that they can live in them. All these examples feature species that massively modify the ecosystem for numerous other organisms, and these are often called *ecosystem engineers*. But importantly, niche construction is present in all species, beyond those engaged in this form of impressive environmental modification. All organisms consume food and emit waste, thereby changing the flow of energy and nutrients in their environments.

A specific form of niche construction is *co-evolution*, of which a prominent form is co-evolutionnary arms races between predators and prey. Imagine for instance an ecosystem with a cheetah and a impala. The impala has a pressure to evolve faster running, as those impalas will survive at higher rates and reproduce more. As the impalas run faster, the cheetahs are also subject to a pressure to evolve faster running. This feedback loop continues to push both species to ever faster speeds until the physiological limits of organisms is reached, as is done in the case of the cheetah who must rest for a very long time after capturing its prey. Another example of these kinds of feedback loops are due to sexual selection. A trait might be desired by individuals of one sex because it is a good indicator of healthy mates. A preference for this trait is evolved, as are exaggerated forms of the trait, and these two co-evolve and form a specific type of feedback loop called a *Fisherian runaway* (Fisher, 1999). A classical example is the extravagant tail of the male peacock. Genes for preference for long tails and genes for long tails co-evolved, expressed respectively in females or males, until the length of the peacock tail of males got close to the limit of what the organism could sustain without endangering its chances of survival. These feedback effects usually result in rapid changes in species, on the evolutionary scale. The duality between stability and feedback mechanisms could explain punctuated equilibria.

Figure 1.8: A cheetah catching an impala. Cheetas can reach up to 90km/h, while impalas can speed at 80km/h. Picture originally from biographic.com.

1.2.2 Cultural evolution

We have thus seen how natural evolution and its processes of variation, transmission and selection lead to the neverending emergence of novel species, for billion of years. We have emphasised the role of feedback processes in creating novel species. When *Homo Sapiens* emerged, with it came a new phenomenon: cultural evolution. The cognitive niche inhabited by humans gave rise to transmission of learned behaviors, variation in these behaviors and their differential propagation, based, among others, on how useful they were to individuals and groups. These are the ingredients for evolutionary processes. The analogy between genetic evolution and cultural evolution was first made by Richard Dawkins (Dawkins, 1976). He proposed that an equivalent to genes in the cultural domain be called memes. Cultural evolution gave rise to cumulative human culture, which is the cause of the wide success of humans as a species.

Humans are especially good examples of a niche constructing species. The increasing sociality of early humans made their environment more uncertain, which in turn created pressures for more complex cognitive processes such as causal, abstract reasoning and theory of mind (representing other individual's mental states), and eventually leading to the emergence of joint and collective agency (Tomasello, 2022). Their arrival on the different continents was concomitant with the disparition of great mammal species such as mammoths and giant sloths, presumably due to large game hunting, creating pressures to evolve new means of sustenance for early *Homo* groups. Agriculture, industry, cities, road systems changed the face of the Earth forever, and has put an intense strain on all ecosystems, triggering the beginning of a mass extinction. Humans construct their own culture and all human ontogeny develops within this built environment, physically and culturally. Some scholars have even investigated the fact that great variations in human psychology could be both the cause and the consequence of rapid technological and cultural changes in Western society (Henrich, 2023), which would be one important example of feedback in cultural evolution.

1.2.3 Intrinsic motivation

The previous sections have underlined how open-endedness in natural and cultural evolution is the result of variation, transmission and selection. In natural evolution, variation comes from random mutations and from genome recombination in sexual reproduction; conservation comes from transmission of the genome to offspring and high-fidelity DNA copy mechanisms, and selection comes from the multiscale dynamics (at genome, cell, organism, or group level) affecting the dissemination of the genome through the generations. In cultural evolution, the mechanism of variation is less well known. Nevertheless there has been recent effort to understand human innovation and the role of human creativity (Enquist et al., 2008; Fogarty et al., 2015; Perry et al., 2021). The mechanisms of transmission include imitation, instruction, emulation. Selection effects include individual and group selection in the natural selection sense, social status, and memes being replaced by other memes. It thus seems that open-ended cultural evolution rests on an as yet mysterious set of creative processes. Uncovering a computational model of how innovation comes to happen, and implementing simple versions of this model should yield an artefact that continually gives rise to novel behaviors, forever. As we are interested in AI systems able to generate interesting novelty forever, the processes giving rise to human innovation seem like a perfect place to start our investigation. There seem to be two sets of mechanisms for invention: the ones displayed by adult humans when they stumble upon a new idea, and the mechanisms used by young children to acquire the traits or their culture as well as autonomously discovered skills.

Humans undergo a very long development phase that has no parallel in known animals; that development being both physical but also cognitive. This cognitive development can understood as a solution to the explore-exploit dilemma (Gopnik, 2020) (see also our discussion in Section 1.1.3): under this children would be equipped with mechanisms for exploration and would switch into exploitation later into adulthood. Adult's exploitation ensures favorable and safe conditions for children's exploration. Children are curious, creative and playful (Chu & Schulz, 2020), and these processes support their intense learning capabilities. Adults that explore and innovate also seem to be curious, driven and playful. An example: mathematics are an engaging game for analytically inclined grown-ups. Similar processes of intrinsic motivation might underlie exploration in children, leading to personal discovery, and exploration in adults, sometimes leading to historical discovery.

Developmental psychology

Developmental psychology studies the construction of human cognitive capacities. One of the puzzling aspects of human cognition is that it both seems very general, with some skills being universal across cultures, while many aspects of cognition such as language or norms are heavily dependent on the culture. How does a child, born in an arbitrary culture, develop both these staggering regularities and at the same time becomes a fully-fledged member of the society it was born in?

The pioneer in developmental psychology was Jean Piaget, who for the first time realized that reasoning capabilities in humans might be better understood if one studies their emergence in children. From the 1920s on, he used revolutionary experimental methods for the time and made numerous contributions to the classification of cognitive function and description of children's behavior. Piaget's lasting conceptual contribution to developmental psychology was the notion of *developmental stages*: functional stages the child would go through that necessarily proceeded in succession because the later stages depended on the earlier stages, both in terms of the cognitive capacities of the child and in terms of the the formative experiences needed for the child to learn and pass to the next stage. An example: infants learn how to crawl and walk by themselves between 12 and 15 months of age and this exposes them perceptually to a whole different world, a world of object and dynamic scenes whereas before they would only have seen static environments and human faces up close.

While Piaget emphasized individual learning through and its stages, another great child psychologist of the early 20th century, Lev Vygotsky, leaned towards a more social theory of development (Vygotsky, 1988). Vygotskian theory was centered around the interaction between the child and a social partner, usually conceived as a mature representative of the culture. The social partner plays with the child and scaffolds her interactions with the world. Vygotsky describes three types of activities: first the ones that are reachable by the child on her own; then the activities unreachable by the child either because they are currently too difficult or because they are outright impossible; and finally the activities that are in the so-called zone of proximal development (ZPD). The latter activities are the one which are feasible but only with the aid of the social partner, which in this way helps the child learn genuinely novel interactions. With the help of this scaffold, the child can then revisit this activity when she is alone, after having gained the ability to do so. Vygotsky applies this kind of scaffolding to language learning as well: first the social partner directs the child's attention and exploration through the use of language, and then the child can, alone, control her behavior by speaking to herself. In a final stage this self-directed language that was first spoken out loud is internalized and transforms into inner speech. This process of internalization is one of the cores of Vygotskian theory, and will be discussed in more detail below in our review of autotelic vygotskian AI (Section 1.2.3).

We end this section by noting that the classic nature-nurture divide is not an accurate way to think about developmental processes. Early stages of development are pretty fixed while later ones are somewhat more flexible, but all stages develop from earlier ones through genetically determined learning mechanisms. A better way to look at development is through the lens of the self-organization of a developmental trajectory which has both genetic and cultural components (Tomasello, 2021).

Intrinsic motivation in humans

Intrinsic motivation, or curiosity, has been a florishing area of study since the beginning of the 1960s; we will not claim that this discussion is exhaustive and we refer the reader to more in-depth reviews of intrinsic motivation, computationally and in psychology (Singh et al., 2004; Oudeyer & Kaplan, 2007b; Kidd & Hayden, 2015a; Ady et al., 2022). Up until the middle of the 20th century the study of behavior and motivation (primarily through the behaviorist perspective of Skinner) had identified a set of classical primary reinforcers, stimuli that allowed the fixation of associations between stimuli and responses. Primary reinforcers give rise to secondary reinforcers, stimuli that become associated with stimuli associated to primary reinforcers. In 1959 White (White, 1959) started to realize that the classical work on motivation as drives (Hull, 1943) (deviations from the homeostatic equilibrium that need to be corrected to keep the organism within its zone of comfort, for instance the drive to satiate hunger, or come back to a comfortable temperature) could not apply to describe exploratory behavior observed in animals and humans. White instead sought to define a specific motivation linked to the efficiency with which the organism was able to bring about changes in its environment. This view was complemented by those of Berlyne (Berlyne, 1960), that investigated the apparent intrinsic drive of children and adults to be drawn to novelty, uncertainty, complexity and surprise. More specifically, Berlyne showed that the hedonic pleasure experienced from novelty would be maximal for intermediate levels of novelty: stimuli that are too novel produce discomfort or confusion, and stimuli that are not novel enough produce boredom. We will find this Goldilocks principle of "just the right amount" later in the work of Csikszentmihalyi on optimal challenge, or the concept of *flow* (Csikszentmihalyi, 1990). According to this theory, people are motivated intrinsically to partake in activities that challenge them optimally, neither to easy not too hard compared to their current skill level.

A recent revival in curiosity research in humans happened in the last 20 years (we refer to Kidd & Hayden (2015b) and Gottlieb & Oudeyer (2018) for comprehensive reviews). Many accounts and experimental paradigms focusing on information-seeking behavior, such as gaze allocation or various bandit paradigms where costs in a task are tradeoff with the opportunity to gain more information, either for instrumental (extrinsic) and non-instrumental (intrinsic) reasons. Some of these experimental paradigms can be tested on humans of any age as well as on nonhuman primates, making them ideal to study intrinsic motivation across species. These rigorous and controlled experimental settings also allows for the scientific study of the neural correlates of this type of curiosity, either through imaging in humans or through invasive techniques like neural recording in monkeys. Related to this recent research development is a new trend in cognitive science, and in developmental cognitive psychology in particular, to model the child as a little scientist making inferences and building hypotheses about the world and then making optimal interventions to resolve any uncertainties; usually this modelling is done within a bayesian inference framework. Even in infants, the orientation towards surprising stimuli is a very strongly established result, robust enough so that the duration of infant gaze can be used as a rigorous measure of violations of expectation of her model.

This recent focus on curiosity as information-seeking motives in adults, children and nonhumans has been very productive and has led to many interesting results. Unfortunately the problems they investigate are still far from the problems of creative curiosity that are a part of the origin of innovation and that would provide us with insight on how to build open-ended creative machines. The experimental settings mentioned in the previous paragraph are usually not close to real-life stimuli, and curiosity can only be directed at a small predefined set of options. Another related, emerging trend is starting to take a closer look at intrinsic motivation in open-ended scenarios (Molinaro & Collins, 2023) and even at children's play (Chu & Schulz, 2020, 2023a,b), which is notoriously difficult to study in the wild. These pioneering studies are starting to allude at distinctive features of playful behavior and fun, such as its non-optimality (taking the longest route to retrieve objects or incurring unnecessary costs when prompted to play, see Rule et al. (2023)). Other recent work has started looking at games through the lens of program synthesis, by looking at the properties of reward programs constructed from asking human participants to invent new games (Davidson et al., 2022).

The landscape of curiosity and intrinsic motivation research is rich and vibrant, and many paradigms coexist in psychology and neuroscience, sometimes radically different. These theories and experiments are useful inspiration for establishing computational models of intrinsic motivation to help us accomplish our goal of novelty-producing machines.

Computational models of intrinsic motivation

In Section 1.1.3 we have discussed the reinforcement learning framework, in particular how adapted it is to the computational implementation of intelligent agents faced with sequential decision-making problems in possibly stochastic, reward delayed environments. RL comes also comes at the rescue for modelling intrinsically-motivated agents. For a given sensorimotor flow (defined by the observation and action of the classical RL control loop) we can compute any scalar function of the observation, action and past state of the agent and RL lets us optimize it. This gives us a design space for our definitions of intrinsic motivation. Many such approaches exist in the AI literature, since this topic first came into preeminence in the early 2000s and was adapting some work of the epigenetic robotics community which we will also survey since they are an important part of the genealogy of ideas used in this thesis. Most of the approaches discussed *will* have at their core the computation of some explicit intrinsic reward and its maximization as part of an RL loop, but we will discuss traditions where the maximization of this metric will be implicit, and it will be indicated as such when it arises. We will also give an mostly non-technical overview without explicit formulas for ease of reading. The categorization we use comes from the seminal classification of Oudeyer & Kaplan (2007b).

Knowledge-based intrinsic motivation The first and perhaps simplest form of intrinsic motivation (IM) are so-called knowledge-based intrinsic motivations. Knowledge-based intrinsic motivations are based on measures of how much the observations fit the agent's world model, by being uncertain, surprising or, on the contrary, expected. For predictive models a classical IM is to use the model prediction error to reward the agent (Pathak et al., 2017); this makes a lot of sense from an information gain perspective since the novel places in the environment that the agent hasn't experienced will have very high prediction error, the model having not been trained on data from these regions. These kinds of intrinsic motivations have many issues: the most prominent one is that the agent supporting them will get attracted to sources of noise since this will have for instance high prediction error. This problem is well-known in the literature and has been referred to the noisy TV problem (Kaplan &

Oudeyer, 2007). One way of dealing with the noisy TV problem is to add more information; for instance through disagreement between models (Pathak et al., 2019). Disagreement in dynamics models allows to tell apart so-called *aleatoric* and *epistemic* causes of uncertainty. Aleatoric uncertainty is the randomness inherent in the environment or simulator (or uncertainty that the agent's models do not have the capacity to resolve), whereas epistemic uncertainty is the uncertainty that comes from the incompleteness in knowledge of the agent (and that could be resolved through further learning). Aleatoric uncertainty should not be seeked out by the agent whereas reducing epistemic uncertainty is the whole point of the drive to acquire information and should be pursued. Another way to solve the noisy TV problem is to go 1st order: use the improvement prediction error or in uncertainty to reward the agent (this is usually called knowledge-based learning progress) (Schmidhuber, 2010). Aleatorically uncertain situations will stay as unpredictable as the agent encounters them more often and will not spark an increase in learning progress, and thus the noisy TV has been avoided.

Another IM that is based on a system's current knowledge is the notion of maximizing novelty. This is in line with psychologists's (and notably Berlyne's) theories on curiosity but interestingly the research tradition most relevant to novelty is the open-endedness community stemming from evolutionary computing. The seminal work here was *novelty search* (Lehman & Stanley, 2011a), and the striking message of this work is that in optimization, when the fitness function is either very sparse or deceptive, direct optimization might do more hard than good, getting agents stuck in dead-ends or not getting any signal to begin with. In that case, it might be better to perform a search guided by novelty only: individuals of the population would be selected based on how close far away they are (in 2d space for the original work), from all the previously evolved agents. By evolving this population in a closed maze some agents actually end up maximizing the original fitness, much better than if this objective had been maximized directly (see Stanley & Lehman (2015) for an extended discussion of these ideas). Novelty search led the way to *quality-diversity methods* (Pugh et al., 2016) that optimize for a diverse population of individuals which are optimal locally with respect to some (independent) quality score. The diversity mechanism in quality-diversity methods often takes the form of maintaining a map of diverse individuals (in some space measuring the dimensions of diversity we care about, and which is usually predefined) (Mouret & Clune, 2015b). The space one measures novelty, as distance, is called the behavioral characterization (BC) space. Building an appropriate BC space is an issue that comes time and time again: the dimensions are usually hand-defined, and this space suffers from the curse of dimensionality.

Novelty has also been leveraged in RL setups to compute intrinsic motivations through count-based exploration methods (Bellemare et al., 2016b). The main idea is to provide an exploration bonus to the agent based on how little is has visited a state, with the maximal reward for completely novel states. The exploration bonus is usually proportional to the inverse square root of the visitation count. Originally designed for discrete MDPs with few states like grid-worlds (Strehl & Littman, 2008), the method has then been extended to continuous and high-dimensional state spaces through density estimation.

As a note, we can remark that many of the approaches in knowledge-based IMs fit in quite nicely with the perspective on information seeking in the huma animal study of intrinsic motivation surveyed earlier, that is focused on surprise (prediction error) and novelty.

Competence-based intrinsic motivation Knowledge-based IMs focus on improving an agent's model of the world. But in general an agent might not need to model the world to effectively act in it (as in the case in model-free RL methods such as the ones we presented earlier in this chapter). If you're a thermostat, you might not need to model the weather to control the temperature

properly. This motivates the development of intrinsic motivation approaches which are based on measures of competence on tasks rather than measures of information gain. *Competence-based intrinsic motivations* are IMs that are are tied to an agent's ability to solve a task. Many such IMs are in particular functions of optimal intermediate challenge or optimal difficulty, recalling notions of flow (Csikszentmihalyi, 1990) and related concepts in psychology (Charms, 1983; Deci & Ryan, 1985), and similar to Berlyne's Goldilocks princple, applied to compentence-based IMs. As with knowledge-based IMs, 0th order metrics can be fooled by randomness, and optimizing for intermediate challenge might mean focusing on non-controllable tasks such as a coin flip. As in the knowledge-based case the solution here is to go first order: use learning progress (LP) and focus on tasks which are highly learnable ().

This line of research introduces the notion of task, or skill, or goal (the terminology might depend on the line of research considered), that must be achieved and on which progress drives motivation. A skill, or goal, is a compression of behavior: there is a representation that uniquely identifies the skill in some space and the agent uses this information to drive behavior that is specific to this skill. We will linger on this notion in more computational depth in our discussion of autotelic learning in Section 1.2.3, but we can already point out that the notion of skills as temporal abstractions has a long history in reinforcement learning through the framework of *options* (Sutton et al., 1999). An option is an abstract action an agent can take once a certain number of preconditions have been met and that uses policy specific to that option until the option's termination conditions have been reached. What's important to note here is that to allow the agent to know which experience is linked to which goal-reaching episode to compute intermediate difficulty or learning progress on a specific skill one does need to partition or label the space of experience into skills. There are multiple ways to do so, either by clustering the skill space directly (IAC (Baranes & Oudeyer, 2013a)) or with unsupervised information-theoretic methods of skill discovery based on skill discriminability (Eysenbach et al., 2018). Competence-based IMs thus require partitioning experience into identifiable zones or clusters, a process we will refer to as building a skill repertoire.

Properties of the sensorimotor flow and empowermentA final type of intrinsic motivation that we'll mention are IMs that depend directly on relations between parts of the sensorimotor flow, without bearing relation to any predetermined model of the world or separation of experience into skills. One example present in the literature on children is the motivation for synchrony of experience (Rochat & Striano, 2000; Gogate & Bahrick, 1998). An important computational representative of this is the *empowerment* objective (Salge et al., 2013). Empowerment is an information-theoretic measure quantifying future optionality: agents motivated by this drive will act in such a way as to maximise their options in the future. More precisely, if one considers the relation between actions and future observations as a noisy transmission channel, maximizing empowerment means acting in such a way as to maximize the capacity of that channel: acting to maximize the information you recover of the action you input into the environment now, n steps later. This IM, while difficult to compute in practice for high-dimensional state spaces, leads to agents that learn to balance a pole (simply because the unstable equilibrium is the point from which all other points can be reached with the least number of actions) or that learn to get out of dead ends in a maze. Recently proposed regularity-based rewards also fall in this paradigm (Sancaktar et al., 2023).

All these computational models of intrinsic motivation provide ways to compute scalars that can be used to reward a reinforcement learning agent towards more interesting, one hopes, parts of the state space. However these metrics do not necessarily lead to the same sort of self-organization of behavior into distinct stages, as young children seem to go to, nor do they necessarily lead to building

an explicit skill repertoire, as discussed above. To perform this we've seen that we might be advised to look at computational models that are explicitly organized around the notion of skills or goals. Springing from a genealogy of ideas coming from the competence-based IM literature is the notion of *autotelic learning*, which does precisely this.

Autotelic learning

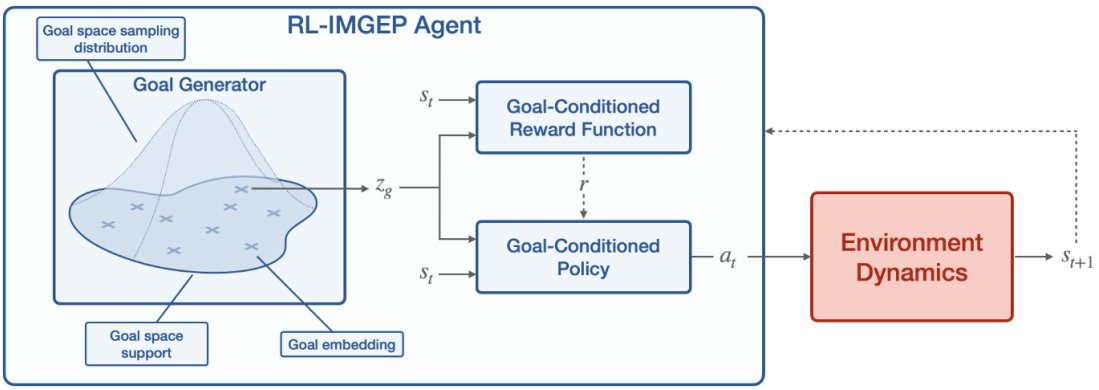


Figure 1.9: An overview of the different modules involved in the autotelic RL framework. The goal sampler samples a goal which then influences the behavior of the policy and how rewards are computed. There is an optional dependency between the experienced trajectories and rewards, and the goal generation process. Image from Colas et al. (2022c)

Autotelic (auto-telos, one's own goals) (Csikszentmihalyi, 1998; Steels, 2004) learning is a form of learning where an agent sets its own goals and tries to achieve them (Schmidhuber, 2011; Herrmann et al., 2022; Colas et al., 2022c). The term comes from Csikszentmihalyi who describes the autotelic personality type as a type of person doing activities for their own sake rather than for an external purpose. By analogy, the autotelic agent engages in self-determined goals for the intrinsic reward that these goals generate. Interestingly, the selection of goals themselves may or may not be governed by an IM based on competence, it can also be random. The high-level description of an autotelic agent is the following: a goal generator $\mathcal{G}$ generates a goal g: the goal might be selected from a list of preexisting goals or sampled from a distribution. This goal is then used to guide (or condition) a policy π (in all this thesis we will use the phrase goal-conditioned policy to speak about policies that should achieve a particular goal). Goal achievement provides learning signal and reward to the policy in the same way that usual task completion is used as reward in extrinsically-motivated RL agents. Originally called intrinsically-motivated goal-exploration processes (Forestier et al., 2022b), autotelic methods incrementally build skill repertoires that can be re-used when training is over by instructing the agent to perform any of the goals it has mastered.

It is important to note that there are two notions of motivation or reward in this framework (in the case of autotelic RL agents): the goal-satisfaction reward R_g and the intrinsic motivation guiding the selection of the goal. Importantly, certain intrinsic motivations might have much better properties than others when selecting goals. An important IM in this regard is the competence-based learning progress (LP) because it specifically helps select the goals on which the agent learns maximally (Baranes & Oudeyer, 2013c; Forestier & Oudeyer, 2016). This steers the agent away from concentrating

on impossible or random tasks, and even more importantly this leads to the self-organization of developmental trajectories where easy goals are the focus of the agent first and harder goals are the focus of the agent later; this starts looking a lot like the developmental stages uncovered by developmental psychology. LP-based goal selection is an example of automatic curriculum learning (ACL: Portelas et al. (2021)), wherein an agent self-organizes a sequence of tasks that helps it learn maximally.

We have presented the autotelic agent as an agent equipped with goal-conditioned policies that define its skill repertoire, but older methods leveraged population-based methods for representing libraries of skills (Baranes & Oudeyer, 2013c). This bears resemblance to novelty-search and quality-diversity methods: a population-based autotelic algorithm evolves a population of skills, selecting them based on a fitness that is given by the IM measure chosen in the design of the algorithm. An important difference still is the goal-targeting aspect of autotelic algorithms: when, for instance in Map-elites, an individual is selected to be mutated, there is no notion of which kind of target individual one should achieve: random variation is simply applied; whereas with autotelic algorithms a goal is targeted and must be reached with the given phenotype. This analogy between quality diversity methods, methods inspired by novelty search, and autotelic methods is especially visible in the go-explore algorithm (Ecoffet et al., 2021). Go-explore was a method designed to overcome hard-exploration RL problems where rewards are particularly sparse or deceptive. The model environment used was the Atari game Montezuma's revenge on which almost no score had been achieved by any of the state-of-the-art Deep RL agents up until then. The idea was to reset to a particular state of the environment and to explore randomly from there, and add the end of the trajectory to the set of resettable states, indexed by a compressed version of the final observation. In essence, this can be viewed as a form of novelty search (one mutates the individuals, which are the set of states one can return to, and saves them if they are sufficiently novel, e.g. if the compressed final observation is different from the ones previously encountered). But one another way to view Go-Explore is that the compressed observations serve the role of goal representations that are sampled by the algorithm as a basis for creating novel outcomes (by further random exploration).

Autotelic agents incrementally build skill repertoires. This was already an important element in the first days of intrinsically motivated reinforcement learning (Singh et al., 2004; Barto, 2012) where skills were acquired in the framework of options. For the goal-conditioned aspect of autotelic learning to work, one can either rely on an explicit collection of skills (as in population-based methods) or on multitask agents implemented through goal-conditioned deep policies. The advantage of this approach is that many skills probably need the same underlying representations, so it makes sense to use a common neural net to compute actions for achieving different goals in the same environment. Pioneering work in multitask reinforcement learning (Schaul et al., 2015a) has demonstrated that the goal representation could be given to the agent's networks. The issue then becomes how to create truly multitask agents that can continually learn novel skills without forgetting the old ones, a problem close to the field of continual learning (French, 1999). This is in general difficult, but leveraging scale, diversity of experience or automatic curriculum learning are good starting points, as is the use of hindsight experience replay (HER) (Andrychowicz et al., 2017). HER ensures that when targeting a goal that is eventually not reached, we can make use of what goal was actually reached to provide signal to the agent: each trajectory is a positive example for the goal corresponding to the last state. Automatic curriculum learning based on Absolute Learning Progress (ALP) (Portelas et al., 2020) is also a way to alleviate catastrophic forgetting, because a task that is forgotten will elicit high ALP and the agent will start focusing on it again (Colas et al., 2019a). Related to multitask agents is the notion of instruction-following; for instance in vision-language navigation (Anderson

et al., 2018b) where agents have to learn to follow language instructions to get to a goal; other very recent examples are instruction-following in language models: language models *are*, after all, multitask learners (Radford et al., 2019), and robust ones that can be finetuned to specifically behave as goal-conditioned agents (Ouyang et al., 2022) whose goal representation lies, in language, alongside the textual behavior. See Section 1.1.5 for more discussion on the links between LLMs, scale and emergent multitask abilities.

A final note on autotelic learning is the degree to which the self-generated goals can be steered by other pressures than intrinsic motivation, especially normative pressures. Capable agents generating their own goals might be scary because they might elude human control, but they need not be. Several perspective works (Colas et al., 2021; Sigaud et al., 2021) outline the possibility that autotelic agent's goal sampling might be influenced by social pressures and norms of what is acceptable and desirable in a given cultural context. This is sometimes called Vygotskian autotelic AI because of the analogy to children in the Vygotskian theory internalizing the social norms of their milieu. Children (and adults!) are autotelic, and their invention of goals is mostly not a problem because it adheres to rules and norms of acceptability and morality.

Creativity: goal imagination

A final ingredient we need in using autotelic AI as a framework for continually inventing new, interesting things is creativity. Creativity is often defined as a perfect blend between novelty and relevance, but this comes back to the question of how to select good IM for our autotelic agents. While one could try to figure out a good metric for creative output, this may be hard to do and the litterature (and parts of this thesis) show there are ways to generate creative goals that do not use such metrics. We will also develop ideas, in this manuscript, about how this mechanism might then be guided by the notion of intrinsic motivation that we decide to use in the end.

Language seems like the perfect candidate for creative generation. Most sentences we utter are completely novel, and yet we understand them immediately. Using words, recombining concepts, one can imagine many things which do not exist in the world (yet), such as rockets to Mars or mythical creatures. These properties are respectively called the *productivity* and *compositionality* of language. Building on ideas from Bruner and Vygotsky about the role of language in creative thinking, Colas et al. (2020b) proposed the Imagine agent, able to form novel goals by recombining parts of goals it had already seen. The goals are learned through linguistic labels for its behavior collected through the feedback of a social partner with an existing knowledge of language (modelling the social partner in Vygotskian theory). Through a hand-defined model of language compositionality, these seed goals are then recombined to form novel goals, and the learned goal-conditioned reward function generalizes out of distribution, which allows the policy to continue training on these goals until they have been mastered and most of the possible goals in language space have been covered by the agent. Using the creative powers of language in this way enables autotelic agents to go from sampling from a predefined, finite set of goals presented to them (Colas et al., 2019a) to imagining and mastering novel goals. We will have more to say on the subject of language and creativity in the experimental contribution chapters, but for now, let us take a step back and recapitulate the different arguments that we have seen.

1.3 Our contributions

Let us step back and recapitulate the discussions in this introductory chapter. We have first presented the history of artificial intelligence in Section 1.1, from its roots with the Perceptron and tic-tac-toe playing machines to the impressive victories of deep RL agents against human champions in today's most demanding games such as Go and StarCraft2; we finished with today's impressive large language models displaying general knowledge, reasoning abilities and in-context-learning. We have seen that this progress has been made possible by the increasing availability of data, compute, the elaboration of problems and benchmarks and the application of general optimization methods to solve them. However, the perspective in AI is often centered on trying to solve problems. If we are interested in machines that create their own problems in the same way that humans do when playing, researching, creating, set themselves new challenges; how could we build such a system?

We have then turned in Section 1.2 to a description of one of the most compelling open-ended processes in the natural world, which is the endless emergence of new forms in nature through natural evolution. We have concluded that speciation occurs, among others, when there are feedback mechanisms between the species and its environment such as co-evolution and niche construction, as well as when population sizes are small. We have examined the link between variation in natural evolution and those in cultural evolution, and have narrowed down on the notion of intrinsic motivation in humans as a driver of cultural variation. We have then shortly discussed the research in intrinsic motivation in humans, and ended with a short survey of computational models of intrinsic motivation, fit for implementation in AI systems. We have ended the section by introducing *autotelic learning* (Colas et al., 2022c): a setting where goal-conditioned agents select their own goals and learn to achieve them, incrementally building their skill repertoire. We finished by outlining the promise of language as a cognitive tool to imagine new, creative goals for open-ended autotelic learning.

It thus seems like previous work studying autotelic agents, using language as goals, might be the perfect inspiration for building open-ended problem-generating systems. The flexibility of language, its abstraction, compositionality and productivity, seem like excellent properties for agents to instruct themselves; however, previous work has only considered simplified, heuristic, models of linguistic creativity. In this thesis we will concern ourselves on how to transfer this setting to open-ended language: how can an agent select a linguistic goal according to an intrinsic motivation measure? How can an agent learn to master an open-ended set of goals, rather than a finite list? What could the mechanisms of creativity demanded by goal invention look like for these agents, and how can they serve the invention of new and interesting problems? How to compute intrinsic motivation measures in this setting? How can these goals be related to the environment? These are some of the questions we will investigate in this thesis.

1.3.1 Towards open-ended linguistic autotelic AI

As progress on these questions, we will introduce six experimental contributions. These will be grouped in three experimental chapters, and we will conclude this manuscript by a final small discussion chapter. Each of the chapters is thematic and includes two experimental studies. We begin each chapter with a one-page summary briefly listing and linking the contents.

Our first chapter concerns itself with **fully-linguistic** autotelic agents. In the introduction we survey the scientific theories of language from a psychological and linguistic perspective, and use this as theoretical grounds to argue that language, in addition to being an ideal way to implement goals in

autotelic agents, also define a perfect environment for agents to explore: in the experiments presented in that chapter we thus focus on text-based games as our language-only environments (Côté et al., 2019). We then present our first experimental contribution, a study of autotelic agents in the complex textworld ScienceWorld (Wang et al., 2022), analyzing which kinds of intrinsic motivations, feedback, and experience replay are conducive to learning in a space of hierarchically nested linguistic goals. We find that constraining the social partner feedback is important, and that selecting goals according to an intermediate difficulty IM is important for learning to master the goal hierarchy. We then build upon this perspective to present our second contribution integrating large language models as a tool to generate novel, open-ended goals and evaluate their success in a textworld, an approach we call LMA3 (Language Model Augmented Autotelic Agents) that is able to discover and master thousands of freeform goals. We measure various diversity metrics over invented goals, demonstrate that the approach is able to master a battery of externally-defined goals, and perform a detailed ablation study of the method. We conclude by outlining the limited nature of the text-based environments we consider, and reflecting on the possibility to extend our results to richer settings.

In our second chapter, we ask ourselves how we could ground language goals to sensorimotor environment. We center on the notion of cognitive object that we review shortly in the first introductory section, and on learning reward functions as a model of the more complex problem of grounding. We then present our third contribution, which is a study on how to best solve one of the simplest problems that arises in learning about objects in a self-supervised way: **learning to recognize and compare spatial configurations of objects**. We study several architectures based on Graph Neural Networks (GNNs) and conclude that the more relational inductive biases (Battaglia et al., 2018a) the architecture has, the better it is able to learn the task. We then segue into our fourth contribution, which studies the problem of learning a **reward function relating spatio-temporal language to trajectories of an agent** and various objects. We define a simplified language with spatio-temporal semantics and demonstrate which architectures are able to learn this correspondence: our main finding there is that the transformer which incorporates the least amount of imposed structure is best, but that aggregating information about the trajectory temporally first is a close second. We conclude on a list of the ingredients to overcome if we are to ground open-ended language goals into the sensorimotor flow of an autotelic agent, beyond the simple templated language cases we have examined.

In our third chapter we tackle the issue we were left with at the end of our first chapter when training autotelic agents in text-based games: the limitation of these environments. In this chapter we propose that programming could be a perfect domain for autotelic agents, because of the truly open nature of code. Building on this insight and on a domain of programming puzzles (Schuster et al., 2021), we introduce our fifth contribution, presenting a study on **algorithms generating an interesting diversity** of programming puzzles. To do so we build on LLMs as puzzle generators as well as an engine to build a semantic representation space within which diversity should be maximized, and call the resulting method ACES (Autotelic Code Exploration through Semantic descriptors). We find that this method is able to generate a higher diversity of programming puzzles on several measures and conclude on the need for a metric linking the generated dataset with a measure of increased programming competence over a given test dataset. Finally, we present recent ongoing work, our sixth contribution. In this work we cast the autotelic framework as a **game between two language-model agents, one inventing goals and one trying to achieve them**; we call this game Codeplay. After presenting first preliminary results on this contribution, we conclude the chapter with a discussion of the numerous threads of research the perspective of code-generating autotelic agents open.

We finally end this thesis with a short conclusion summarizing our and linking our results.

1.3.2 List of contributions

The experimental contributions are associated with the following papers (asterisks denote equal contribution):

- **Deep sets for generalization in rl**; Tristan Karch, Cédric Colas, Laetitia Teodorescu, Clément Moulin-Frier, Pierre-Yves Oudeyer; *Beyond Tabula-Rasa in RL Workshop at ICML 2020* (not covered here);
- **SpatialSim: Recognizing Spatial Configurations of Objects with Graph Neural Networks**; Laetitia Teodorescu, Katja Hofmann, Pierre-Yves Oudeyer; *Frontiers in Artificial Intelligence*;
- **Grounding Spatio-Temporal Language with Transformers**; Tristan Karch*, Laetitia Teodorescu*, Katja Hofmann, Clément Moulin-Frier, Pierre-Yves Oudeyer; *Neural Information-Processing Systems, 2021*
- **A Song of Ice and Fire: Analyzing Textual Autotelic Agents in ScienceWorld**; Laetitia Teodorescu, Eric Yuan, Marc-Alexandre Côté, Pierre-Yves Oudeyer; *preprint*;
- **Augmenting Autotelic Agents with Large Language Models**; Cédric Colas*, Laetitia Teodorescu*, Pierre-Yves Oudeyer, Eric Yuan, Marc-Alexandre Côté; *Conference on LifeLong Learning Agents, 2023*;
- **ACES: generating diverse programming puzzles with autotelic language models and semantic descriptors**; Julien Pourcel*, Cédric Colas*, Pierre-Yves Oudeyer, Laetitia Teodorescu; *Under review for the International Conference on Learning Representations, 2024*;
- **Codeplay: Autotelic Learning through Collaborative Self-Play in Programming Environments**; Laetitia Teodorescu, Cédric Colas; Thomas Carta, Matthew Bowers, Pierre-Yves Oudeyer, *In preparation, prensented at the Intrinsically-Motivated Open-Ended Learning Conference, 2023)*;

Chapter 2

Language-based Autotelic Agents

Preamble to the chapter

This first experimental chapter will be about autotelic agents in text-based games. We begin by a detailed background (Section 2.1) on theories of language in cognitive psychology and linguistics, trying to tease out what we know about the open-endedness of language. We then move on (Section 2.2) where we present a study of an autotelic agent using language goals in a challenging text-based game, where observations and actions are linguistic. We define a hierarchy of language goals the agent can master, and learning to associate goals and behavior happens through the feedback of a social partner. In ablation studied we establish that goal-selection based on intermediate competence is instrumental in learning to master the goal hierarchy, as is filtered feedback from the social partner. We then move on to our second study (Section 2.3) presenting LMA3, a method integrating Large Language Models (LLMs) into the autotelic architecture for open-ended goal generation. We demonstrate that our agent is able to discover thousands of goals and measure the diversity of these goals in various ablations. We conclude this chapter by outlining the promises of large models for autotelic learning, especially in more open-ended environments. The Sections 2.2 and 2.2 reuse material from the following contributions:

- **A Song of Ice and Fire: Analyzing Textual Autotelic Agents in ScienceWorld**; Laetitia Teodorescu, Eric Yuan, Marc-Alexandre Côté, Pierre-Yves Oudeyer; *preprint*;
  This collaboration with Marc-Alexandre Côté and Eric Yuan was born of a visit to Microsoft Research Montréal, and followed some unpublished experiments in non-textual environments.
- **Augmenting Autotelic Agents with Large Language Models**; Cédric Colas*, Laetitia Teodorescu*, Pierre-Yves Oudeyer, Eric Yuan, Marc-Alexandre Côté; *Conference on LifeLong Learning Agents, 2023*;
  This collaboration was elaborated while the previous one was still ongoing following, a discussion with Cédric Colas and reflexion on how to use language models in linguistic autotelic agents. Cédric and Laetitia devised and ran the experiments, Cédric ran the analysis and wrote the paper, Laetitia drew the figures and edited the paper.

2.1 Introduction to the chapter

In this section, our aim will be to first motivate the use of language-augmented (or linguistic) autotelic agents, and then present two concrete implementations that answer the following questions: how can we efficiently learn skills in large, linguistic spaces? How can we efficiently generate goals for open-ended linguistic agents? How can linguistic priors of relevance or interestingness influence the learning of the agent? Can these agents truly be open-ended, and if not what is missing?

To answer these questions we must first motivate the use of language as a relevant space and medium within which to operate. On first consideration this might not seem obvious: the aim of building agents that can behave in the world would naturally orient us towards robotics, or simulated, complex, physically-based environments. After all, the most prominent example of an open-ended process, natural evolution, happens in complex physical environments. On of its main driver in multi-cellular organisms is the complex changes in form brought about by mutations in a select few genes that govern the unfolding morphogenesis in the developing organism (Carroll, 2006b). Can language support the same level of complexity and evolution as these biological processes?

We will argue in this background paragraph that it can, due it its unique structure. Furthermore, we will argue that language is the perfect support for the expression of goals in autotelic agents. To see this, we will first examine the unique combinatorial properties of language, through some of the historically important accounts of language in linguistics and machine learning. We will then turn to the links between language and cognition to support the relevance of using language goals as the main agentive mechanism for our autotelic agents. And finally we will end this background section by considering a classical objection to operating only in symbolic spaces: the symbol grounding problem. We will see that the issue of how to link the symbols of language to the external world is solvable with the approach of *functional grounding*. Throughout the section we will complement theories and arguments from cognitive science and linguistics with examples and implementations in AI and machine learning. Thus, since language is the ideal medium for goal expression and imagination, and since connecting symbols to low-level sensory data is not completely necessary (nor advisable), we will be ready to present our contributions involving language-using agents in language worlds.

2.1.1 Language: symbols, abstraction, and syntax

In this section, we will review what is known about language in humans. What are its properties? How can its richness be described? We will see that human language forms a *symbol system*, with, importantly, complex networks of links between symbols that can be meaningfully recombined. This is at the base of the infinite expressive power of language and the focus of linguistic theories which we will shortly review here. Thus, language defines an infinite space to explore.

What is a symbol?

Many animals produce utterances such as cries or songs to communicate, and communication exists between cells in a body through chemical and electro-physiological signals. But of all the forms of communication that have arisen through natural selection, human language is distinctive. It seems to allow for the expression of arbitrary meaning, adapting to the infinitely varied environments and situations that humans might want to express in varied environments and in their particular culture. Thus it cannot be hardwired and must be learned.

This feat of adaptive communication is achieved through the use and manipulation of representative signs: representative units, in an arbitrary modality, coupled with a referent (the thing in the outside world the sign designates), the relation being valid for a given interpretant.

An illuminating account on the properties of signs is the one put forth by Charles Sanders Peirce and refined by Terrence Deacon and Andreas Nieder (Deacon, 2012; Nieder, 2009). They divide signs into three broad, hierarchical, categories:

- **Iconic signs**: these signs bear a structural, non-arbitrary resemblance to their referents. For instance, a iconic counting system would use a number N of marks to designate N objects. An iconic writing system would use simplified drawing of a house to refer to the actual building within which one lives. Icons implement a one-way reference to their referents: the sign for the house designates the house, but there is no sense in which the house itself is linked to this particular icon.
- **Indices**: these signs are higher on the ladder of abstraction: the relation between sign and referent is arbitrary but still unidirectional.
- **Symbols**: this is the most complex and powerful sign system. In addition to bearing arbitrary resemblance to their referents, symbols are bidirectional. This means that the symbol user can always translate from the symbol to the referrent and vice versa. As an example, the preferred word for the building in which I live will always be "house" in the cultural group that shares the English language. But even more distinctive than bidirectionality is the web of relations that symbols support. Many symbols can be related in pairs or tuples, and the structure of these relationships mirror something about the structure of the referents themselves. When I say that "the cat eats the dog", the relation between the symbols say something about the relation of the entities these symbols refer to. Moreover, this relation has a *syntax*: the position of the words are underlied by certain rules, and changing this order either leads to invalid sentences or refers to a different states of affairs. If I say "eats cat the the dog", the situation referred to by this sentence might not be understood; in, on the other hand, I use the proposition "the dog eats the cat", the order of word-symbol differs and the locutor will refer to a different state of affairs than with the first sentence. The propositions built from several symbols through syntax can themselves be combined to form even larger constructs such as an argument, essay or book.

Figure 2.1 provides an illustration of the three levels of sign systems. Human language is a symbolic system. Hauser, Chomsky and Fitch have argued that the recursive, combinatorial aspect of language is what differentiates it from all other animal communication systems (Hauser et al., 2002). Indeed, while many attempts have been made to teach human language to animals such as parrots, dogs or nonhuman primates (so-called enculturated apes), none have achieved to elicit in them a linguistic capability on par with our own. Looking at how animal communication and how far they can go in learning something like human language can help illuminate what is so special about uniquely human language.

No other animal is symbolic

Even if animals don't naturally develop language in a natural environment, could they be taught? Could they have the faculty of language if they didn't need to do the hard work of coming up with it? Some animals have been taught hundreds of words by humans and can adapt their behavior accordingly. Great apes can learn basic visual systems to communicate (Premack & Premack, 1972; ASANO et al., 1982) as well as hundreds of signs from American Sign Language (Gardner & Gardner,

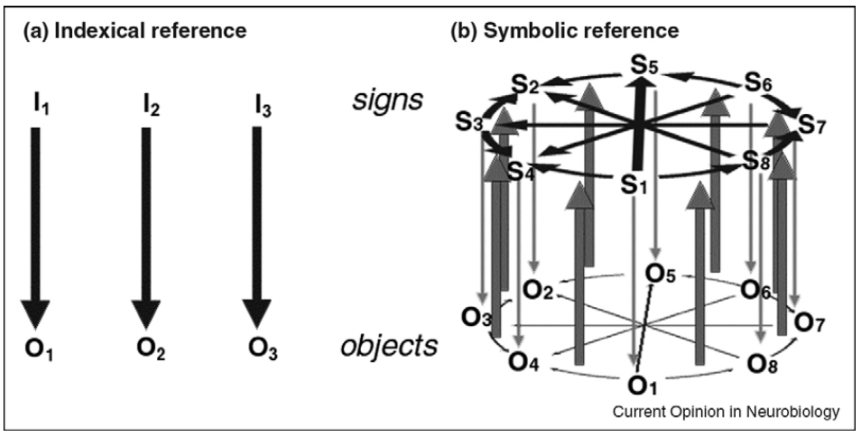


Figure 2.1: Indexical reference versus symbolic reference. In indexical reference (a), the sign and their referents entertain a one-way relationship, and all these relations are independent of each other. Symbol systems (b), on the other hand, rest on networks of sign-sign relations (top black arrows). Relations between signs mirror the relationships between objects. Figure from (Nieder, 2009).

1969). Bonobos also exhibit the capacity for learning sign systems (Savage-Rumbaugh et al., 1986; Brakke & Savage-Rumbaugh, 1995). Farther genetically from humans, a dog called Rico has been taught over 200 words (Kaminski et al., 2004) and has even been shown to learn novel words rapidly, exhibiting what developmental psychologists have called "fast-mapping" in children (Carey & Bartlett, 1978). Dolphins have been taught a few dozen arbitrary signs (Richards et al., 1984). Parrots have also demonstrated the ability to acquire a rudimentary referential system of up to 50 signs (Pepperberg, 2000).

However, even when learning indexical sign systems nonhuman animals struggle with reversibility, syntax and recursion. Rhesus monkeys and baboons fail massively in tests of reversibility of the equivalence relation (Sidman et al., 1982). In a chimpanzee raised in captivity, there is little generalization between languages learned in productive (producing language) and receptive (language understanding) modes (Kojima, 1984). Recent work has investigated whether artificially inducing categories in guinea monkeys could help them learn the symmetry relation and found only weak effect, underlining the inherent difficulty of equivalence relationships in animals (Medam et al., 2016). Reversibility seems to be a unique feature of human language.

Syntax and recursion, believed by Hauser, Chomsky and Fitch to be at the core of the human faculty for language, are not exhibited by nonhuman animals. A long and comprehensive study with a chimpanzee, taught early in development a sign language of over 130 words and humorously named Nim Chimpsky, found that Nim exhibited basic phrases such as "me hug cat" (Terrace et al., 1979). However, in an extensive study of the chimp's utterances the authors failed to find any evidence of symbol recombination to form new concepts, suggesting Nim's use of words is mainly indexical; the combinations he does use are limited in the number of underlying concepts they represent suggesting lexical rather than semantic use. Other research has surveyed inability to form categories in rhesus monkeys (Smith et al., 2004). It thus seems nonhuman animals are unable to use fully-formed symbol systems as humans do, stopping for the most part at the indexical level of sign use.

Compositionality and systematicity

How do we account and describe the uniqueness of humans language? Noam Chomsky notes, quoting Wilhelm von Humboldt, language's ability to make *infinite use of finite means*. The number of letters, morphemes (phonetic subcomponents of words) or words in any human language is limited, but we are able to flexibly recombine them to create novel sentences that any locutor of the language will understand without any further explanation or training. Again, this is a feat completely out of the reach of all other animals: the animals trained for cognitive tasks (including some surveyed above) have to be reinforced for typically thousands of trials before systematically succeeding at the task. There are two main properties of language that underlie this fascinating multiplicity of possible sentences and meanings. The first is the productivity of language: the fact that words and phrases can be stitched together to form novel constructs. The second one is its systematicity: the meanings of complex sentences can be understood if one understands the meanings of its constituents and the way they are composed together. If I understand "the cat is awake" and "the dog is asleep", systematicity ensures I will understand the meaning of "the dog is awake".

The productivity of language, as seen by linguists

Historically, the main theoretical account shedding light on the productivity of language has been the theory of generative grammars famously put forward by Noam Chomsky, starting the field of computational linguistics (Chomsky, 1965). In the generative grammar account of language, linguistic form emerges from the repeated, recursive application of a finite set of production rules, enabling the progressive construction of a sentence by replacing certain constituents with a compound of other constituents to which the same set of rules apply, so on and so forth until the production of so-called terminals, that are the words of the language and to which no further rules can be applied (see Figure **??** for an example). The role of the generative linguist is then to derive for each language the lexicon, or list of terminal words, as well as the list of production rules. Both the form and the semantics of sentences are derived, in this account, from which production rules have been applied and in which order. This can be represented as a syntax tree describing the structure of the sentence.

Chomsky's approach has been influential in linguistics, and his theory has in particular been successful in providing an explanation both for the recursive structure of sentences and for the rapid acquisition of language by children. This latter issue, dubbed the problem of the "poverty of stimulus" (first discussed as a counterargument to B.F. Skinner's behaviorist theories of language acquisition (Chomsky, 1959)), is that children are not exposed to enough data to learn language from stimulus only. This is taken to imply the existence of a form of Universal Grammar (UG) of which any language's particular grammar is an instance with specific parameters (for instance the position of the verb in the $V \rightarrow VNP$ rule differs in English and Japanese). Learning language is then, according to this account, learning the lexicon as well as learning the specific parameters that define the language in the UG. In this framework, the work of linguists has then been to describe this parametric UG and map out languages to their specific parameters, a research program that is still ongoing (Chomsky, 1995; Cinque & Rizzi, 2012).

This syntax-first approach to language has received criticism in the community of linguists, for its inability to account for non-systematic phrases like "let alone", idiomatic expressions whose meaning is conventional and not deductible from the meaning of its constituents (Fillmore et al., 1988). To account for these cases, authors like Fillmore and Lakoff built the notion of construction grammar (Goldberg, 1995). This approach to linguistics centers on the notion of the linguistic (**?**),

which can be anything from a morpheme to a word to an idiom to a syntactic structure, removing the syntax-lexicon divide that exists in theories of generative grammar. Constructions function like abstract, sometimes partially-filled linguistic patterns whose meaning is in part derived from the construction itself and in part from its constituents. "Let alone", for instance, is a construction that is completely filled and has no additional slots for other constructions, but others like "The more X, the more Y" have to be completed for them to signify anything. Construction grammars provides a set of templates from which the whole language can be built. By recursively filling out constructions within constructions, they provide an account of the productivity of language; however, contrary to generative grammar approaches, they eschew complete systematicity: at least part of the meaning of a complex is determined by the construction itself and not only constituents. Lakoff in particular has argued for this approach because it accounts for the creative uses of language through metaphor: applying the same construction to a different set of objects. Construction grammar provides a convincing account of the creativity of language: the ability to create novel sentences from known ones, through the application of known constructions to known words, in novel combinations. It also provides a convenient unit of selection (akin to the gene or the individual in genetics) for accounts of the evolution of language (Kirby, 2013).

Another important development in theories of language that go against the rule-based, categorical approach of generative grammars are statistical models of language (see Section 1.1.5 for an introduction). Interestingly, during the same time that statistical approaches to language learning were becoming popular in computer science and engineering for natural language processing tasks, there has been continued debate on whether this class of models could accurately model human use of language and language acquisition (Norvig, 2011). In Norvig's analysis, linguists continue to use grammar-based models with few parameters because of the nature of scientific explanation as understood by language scholars. Statistical models with billion of parameters are deemed by most linguists to be a poor scientific explanation of natural language because they do not yield to the precise mathematical analysis that has underlied progress in other sciences such as physics in the 20th century, as formal languages do. However, since the emergence of statistical language learning methods in computational linguistics, some scholars have argued that these provide an accurate model of language in humans (Elman, 1995), and the contemporary iteration of this trend is investigators linking the activations in today's large language models to activations in the human brain when exposed to the same linguistic stimulus (Caucheteux & King, 2022) (it is interesting to note that this latter trend comes from neuroscience, which as a science is less reluctant do deal with systems of unimaginable complexity, rather than cognitive science which has a preference for well-defined, independent modules with a function that is easy to describe). Statistical modelling approaches to language also have an account of the productivity and systematicity of language: productivity is given by the string structure of language which always allows for adding a new word; and systematicity comes from similarities in surface-level linguistic patterns which can be picked up with a powerful enough model, from data alone, and with little a priori knowledge.

We arrive at the picture of human language as a communication system that has no equivalent in the natural world. Its locutors can create and understand novel sentences zero-shot, using already-known components. Some theories focus on syntactic production rules as an explanation mechanism, others with linguistic constructions, others still with statistical regularities (like constructions, learned iteratively and transmitted through culture, and subject to evolution (Kirby et al., 2014)). These accounts give us a hint of how the creative powers of language emerge to express infinite meanings from finite constituents, and how language evolves to adapt to the specific configurations of human's natural and social environments.

Influence of theories of language in machines

Theories of language in humans have had an immense influence on the development of programming languages. The use of Chomskian syntax to describe programming languages was pioneered by John W. Backus, who was a programming language designer at IBM in the 1950s and was aware of Chomsky's work on natural language. He built a "metalanguage" for describing the syntax of any programming language (Backus, 1959), and this innovation was subsequently incorporated in ALGOL. The notation, later dubbed Backus Normal Form, or Backus-Naur Form (BNF), defines a framework for describing context-free grammars and is now incorporated in the specification of almost all modern programming languages.

Construction grammar has also been used for implementing language in machines, albeit at a smaller scale and mostly for academic research on on the emergence and evolution of language. This was the project of fluid construction grammar (or FCG) (Steels, 2011). The system is bidirectional and can be use for production to parsing, and the semantic network obtained from parsing an utterance can additionally be used to drive a robot's actions.

Widespread success has been achieved for natural language processing and generation through statistical methods, and while this framework is not widely used for scientific accounts of natural language in linguistics and has limited influence in cognitive science as an explanation of language representations (in its billion-parameter form), it has gained more and more momentum in engineering applications, from the first working language models (Bengio et al., 2001; Sutskever et al., 2014) operating through long-short-term memory (LSTMs) (Hochreiter & Schmidhuber, 1997) units to the more recent Transformer based Large Language Models (LLMs) (Vaswani et al., 2017; Radford & Narasimhan, 2018; Radford et al., 2019; Brown et al., 2020). Today the overwhelming majority of applications processing language are based on statistical methods and learning, with little prior linguistic knowledge coming into play: a decoupling of scientific understanding and engineering of sorts (Sutton, 2019).

However, even as these methods came to preeminence, it was argued that they lacked the generalization ability of purely symbolic methods, in particular the systematicity property outlined above (also called systematic generalization (Bahdanau et al., 2018; Andreas, 2019)). (Bahdanau et al., 2018) in particular have argued for modular, composable architectures (so that the structure of the model reflects the structure of the data), possibly with strong priors and regularizers. Neuro-symbolic methods, that combine learning from data with flexible recombination of subcomponents, have been proposed to fill this gap (Andreas et al., 2016; Santoro et al., 2017c; Hudson & Manning, 2019). They ususally require less data and do generalize better outside of distribution. However the sheer scale of LLM training and the variety of their data make them pick up regularities that allow them to extrapolate beyond their training data, displaying systematic generalisation (for instance through context learning); neuro-symbolic like those cited before models are not needed to build general language learners. However, a new wave of methods that are neuro-symbolic in essence (or logo-symbolic maybe) have emerged recently based on the successes of LLMs. There methods do not compose at the level of architecture bit at the level of generated text, allowing to define LLM-based programs that operate with natural language input and output (Wei et al., 2022b; Dohan et al., 2022; Schuurmans, 2023; Yao et al., 2023).

Given the recent success of ML-based NLP methods based on Transformers, there has been continued effort in investigating whether the representations and procedures used in classical NLP such as syntax trees, parts-of-speech tagging, or coreference resolution are captured in model internals:

this would support the view that while we cannot build effective machines based on exact linguistic principles, we can still build models that learn those directly from data. These issues have been investigated at length using the BERT masked language model (Devlin et al., 2018) (the field at large has been nicknamed BERTology (Rogers et al., 2021)). This line of research does show some support for the fact that under the hood, these big models recover some of the linguistic notions cited earlier. There is some evidence that the token encodings themselves bear some of the syntactic information (Vilares et al., 2020; Kim et al., 2020; Rosa & Mareček, 2019). Some authors have argued that at least some of the elements of the classical NLP pipeline, including syntax trees, are rediscovered by BERT (Tenney et al., 2019); and some of this information is encoded in the attention weights (Clark et al., 2019). However, even is the temptation is great to treat attention weights in a model as a causal explanation of the produced outputs, some authors have warned that attention is not explanation, and some models for some tasks do not rely particularly on the attention matrix to perform their predictions, invalidating it as an interpretable explanation of the model's behavior (Jain & Wallace, 2019).

Conclusion

Of all Nature's communication systems, only human language has the amazing ability to adapt to virtually infinite situations, easily expressing novel meaning never conceived before by its locutors. Language is a reversible symbol system that is productive (infinite possible sentences) and systematic (the meaning of novel sentence derives from its constituents). No other animal has the faculty of language (but some prerequisites can be found in nonhuman animals). and lastly, different formal linguistic accounts giving different weight to the constituents (as in Chomskian linguistics) versus the structures themselves (as in construction grammars).

Overall, formal accounts of the structure and properties of language have been adopted to build rigorous and exact systems where the semantics are fixed and unambiguous: in formal languages such as programming languages or proof systems. Where fluid natural language understanding and production is needed, statistical methods, in particular machine learning, in particular huge unstructured networks seem to fare best. These systems have become almost as black box as the human brain itself: indicating that language is a complex system with emergent properties whose description cannot fit in a simple, analytically solvable formula. Some structures traditionally studied by linguists do nevertheless seem to emerge from unsupervised learning from form, such as syntactic structure; and applications combining LLMs with structured reasoning have achieved widespread success.

2.1.2 Language, reasoning and thought

Language and thought in humans

After surveying the form of language in humans and in our best artificial systems, we will now turn to an account of the links between language and cognition. The cognitive revolution in psychology was built around the analogy of mind as a computer. Mind, like the computer, takes in inputs from the external environment, performs some internal computation, and produces an output through end-effectors. It is the internal computation defined in the cognitive analogy that we will refer to as "thought" or cognition in the following section. Now that we have seen what makes natural

language so unique in its form and use, what seems to supports its productivity and systematicity, we can begun reviewing and understanding the links between language and cognition in humans.

The nature of cognition itself is poorly understood, but one prominent theory in the history of cognitive science is Jerry Fodor's notion of a language of thought (Fodor, 1979; Rescorla, 2019; Quilty-Dunn et al., 2022). According to Fodor, complex mental operations all happen in a common currency or language, sometimes called *mentalese*, that allows communication between different modules and manipulation of mental symbols. The language of thought hypothesis might have been inevitable in all cognitive accounts of the mind, since these accounts all model mental operations as operations on computers, which, as we have seen, manipulate symbols according to a formal language. Thought as computation loops back to the linguistic origins of computation. Arguments supporting the existence of a language of thought are familiar by now: thoughts are productive and systematic, as language is: so they must be expressed with a medium that has the same properties. Additionally, if the mind is seen as a collection of specialized modules as well as higher level collaboration between modules, some common code must be used to transfer between modules, for instance from an auditory perception to visual imagery, if someone asks me to picture the color of a mango. But even if there is something like a language of thought, surely not all operations of the brain, including subliminal, low-level and unconscious operations can have or in effect need productivity, systematicity, or to be expressed in a universally-accepted code. Where do we need then to postulate a language of thought? Kahneman (Kahneman, 2011) famously divides cognition into 2 types: system 1 cognition that is immediate, fast, imprecise, parallel and intuitive, and system 2 cognition which is slow, deliberate, serial, accurate and self-reflexive. The language of thought might describe system 2 cognition, since this is in the mind the locus of productivity, systematicity and universal representations. But how does natural language relate to cognition, the language of thought, system 1 and system 2 cognition?

The study of linguistic relativism might shed some light on these issues. In the first half of the 20th century, Benjamin Lee Whorf, an American linguist (and fire prevention engineer) articulated the idea of linguistic relativism: the fact that one's thoughts are influenced by the language one speaks. Under this hypothesis, different languages, because they differ in their syntactic structure or their categories would support different cognitive abilities, and locutors of a language which lacks the word for a specific concept would have difficulty thinking about it, if it is not outright impossible. This proposition has come to be known as the Sapir-Whorf thesis. Authors came to distinguish two forms: the strong form where two speakers speaking two different languages would have radically different inner worlds, and a weak form where the specific language has some influence but it stays non radical. This thesis, has been widely discussed in cognitive linguistics, anthropology and psychology. Vygostsy, for instance, believed that a parent's native language heavily influenced the concepts the child would learn first (Vygotsky, 1965). In recent years the scientific community reached a consensus: for perceptual (system 1) processes, the weak form holds (Lupyan et al., 2020): giving for instance locutors of languages with 2 words for blue a slight but significant advantage in color recognition in the appropriate hue; and overall language biases us to perceive in a more categorical way.

However, acquiring a language at all, any language, is vital for humans' normal cognitive development. Deaf children born in non-signing families, never learning a language, fall behind in non-linguistic skills such as math and theory of mind: presumably because mathematical skills such as counting and parsing an equation require something like language, and that complex reasoning about other's hidden motives also involve language explanation. The effect of losing language later in life (as in the case of global aphasia), while significant, seems less dramatic: aphasics seem to retain some theory of mind and mathematics. Maybe language can be understood as a scaffold for the mind,

where learning a language, any language, no matter its syntax, exposes the learner to concepts that have been forged through cultural evolution and that would be prohibitively expensive to discover on one's own. Furthermore, there are some concepts that would be unattainable at all without the abstractions and categories delimited by language. How to understand quantum mechanics, or art history, or literature without language? It seems that for higher-order, abstract concepts, the scaffold of language must hold always.

Beyond the influence of language on system 1 cognition and in learning concepts is the overwhelming importance of language mastery in multi-step reasoning, planning, executive function, and creativity. Authors developing this supra-communicative view of language (Clark, 1996) focus not only on simple cognitive skills such as perception, counting, spatial reasoning and the like, but also on planning, self-regulation, strategic reflection, deliberative thinking, creative imagination. According to this view, this high-level influence of language on thought is either due to the fact language is the representational format within which our thoughts are expressed (Carruthers, 1998) (so language of thought and actual language are one and the same), or because it reprograms the brain from a parallel machine to a serial one (Dennett, 1991), or because it behaves as artifact that reifies symbols that can then be manipulated and attended to in mental operations, simplifying our computational operations (Clark, 1996). Vygostky (Vygotsky, 1965) proposes that as young children are exposed to language in the context of an interaction with a social partner (an adult parent or carer), they learn to imitate the language that is directed to them. In self-directed activities, children then reproduce the same sort of instructions and directives, but to themselves, out loud. After a while the child internalizes the speech they used to produce out loud, so they can use this internal monologue to plan and reason; this process only gets refined with the use of written notes, calendars, to-do lists in adults. Tomasello goes further and argues that while children need to be scaffolded by dyadic interaction with an adult social partner early in their development, the role of peers is also of supreme importance (Tomasello et al., 1993; Tomasello, 2021). By three years of age, when they begin to learn complex language (and form a sense of continuous self that persists through time (Gopnik, 2010)) they also frequently interact linguistically with peers, and this has a different influence on their cognitive development and their ability to become fully-fledged members of their culture. Through interaction with their peers, children learn moral and cultural norms: what members of one's culture can and cannot do, how things should be done, how one sees and is seen by others in relation to these norms. Deviations are met with linguistic, normative protest by peers, and similar to the process in the child-adult dyad these normative linguistic interactions are internalized by the child to form a basis of their morality, culturally-dependent executive regulation, self-evaluation, and self-criticsm On this Vygotskian-inspired view of the supra-communicative view of language, reasoning is a conversation with oneself. Creative imagination also seems to be underpinned by something like language: imagination relies on the same episodic constructive processes as episodic memory (Schacter, 2012), and it has been argued recently that these processes could be underlied by something like a language of thought (Mahr & Schacter, 2023).

The influence of theories of language in machines

Thus we see that language and thought are inextricably linked: language shapes our perceptions, allows us to construct ever-higher abstractions and generalizations, supports reasoning, planning and creativity. Interestingly, this structuring role of language also seems to have its equivalent in our AI systems.

First, language supports abstraction in agent's representations. For instance, (Lampinen et al., 2022) have demonstrated that using an auxiliary task based on generating explanations for why the agent chose a particular solution to the task, while training, allowed it to form representations of higher quality, more abstract and better indicative of the causal and relational nature of the problem. as opposed to simply learning on the task. The abstractions created by language can also help agents explore in a more abstract space, thus bypassing the noisy TV problem for agents with novelty-based rewards (Mu et al., 2022). Linguistic abstractions learned can be flexibly recombined, like language, to form higher-level behaviors using hierarchical RL (Jiang et al., 2019). (Luketina et al., 2019), in their extensive survey on language-augmented RL, highlight linguistic tasks as auxiliary objectives for RL agents that generalize.

Second, language as a reasoning medium has been extensively studied very recently, notably since the introduction of the Chain of Thought (CoT) prompting technique (Wei et al., 2022b). CoT prompting involves guiding the language model to produce a reasoning trace before solving the task, traditionally asking it to sequentially reason about the problem at hand. The LLM then uses this reasoning to produce an answer, improving its performance. LLMs can even bootstrap their own reasoning by generating more examples of reasoning traces based on a few data points, and finetuning themselves on that, an example of self-taught reasoning (Zelikman et al., 2022). Linear reasoning traces have then be extended, for instance with the language model cascade framework (Dohan et al., 2022) where a probabilistic program dictating the control flow of successive language model calls are derived, or in the Tree of Thought approach where instead of a chain of reasoning a whole tree is kept in memory and explored to yield the best answers (Yao et al., 2023). In a way, approaches similar to classical planning approaches are starting to emerge in AI, all supported by natural-language based reasoning.

Third, language has an important role in creativity and imagination in goal-conditioned agents. The seminal work in this regard is the IMAGINE agent of (Colas et al., 2020a). Embracing the cognitive tool view of language, they train an RL agent conditioned on linguistic goals, in interaction with a social partner giving the agent descriptive, hindsight linguistic feedback. In an imagination phase, the agent then uses a template-based approach, similar in spirit to construction grammar, to create new goals based on existing ones. The generalization ability of the object-based reward function allows the policy to master imagined goals outside of its distribution. Thus this agent incrementally explores a larger and larger set of behavior until it has learned everything that can possibly be learned in its playground environment.

Thinking back on the theories of the interactions between language and thought, we can see that the model of (Carruthers, 1998) holds when talking about AI: in the LLM-powered reasoning methods mentioned above thought is at least partly done using language representations themselves as a medium. (Dennett, 1991)'s view is true as well: learning to reason, in all those language-based systems, does implement a serial machine onto a massively parallel computational substrate, namely large neural networks.

Conclusion

We thus have seen that language has a tremendous influence on cognition in humans, and augment and make possible abstract, compositional representations, planning and reasoning, and imagination in machines. It is thus important, to implement general autonomous agents whose capacities rival our own, to allow them to express their goals in language, operate on them using language, and use

language to imagine new, creative goals based on the ones already mastered.

2.1.3 Language, grounding, and acting in the world

Since we now have good reasons to believe that language is a perfect medium for internal deliberations, interna goals and plans, in agents, we must ask ourselves how it is that the symbols these agents manipulate can refer to anything in the real world and not to other meaningless symbols. This has been called the symbol grounding problem by Harnad (Harnad, 1990). In this influential article Harnad first delineates the properties of symbol systems and most notably their compositionality, systematicity and the fact that the operation (or syntactic rules) of the symbol system only depends on the form of the symbol themselves, not their meaning. Harnad then asks how symbols in such an isolated system can acquire any meaning. He compares this situation to trying to learn Chinese as a first language while only having access to a Chinese/Chinese dictionary: while trying to learn what a particular word means, the only thing you would have access to are other strings of meaningless words. Would it be possible to learn anything?

To solve symbol grounding, Harnad proposes a hybrid system. At the time, symbolic AI was the dominant paradigm and neural networks, or connectionist approaches, were less effective and less used. Harnad argues that one could (1) ground some iconic and categorical (indexical?) representations in base perception, and afterwards (2) ground symbols in these tokens. He suggested that one could complement symbol systems with a connectionist approach to perform (1), and this double grounding would form the basis for meaning in the hybrid symbol system.

Interestingly, this is something that has eventually happened in the field of Machine Learning. In image perception the advent of multimodal methods such as CLIP exactly implement a representation system (the internal distributed representations, or latent vectors, of CLIP) that categorizes visual inputs. The conversion from image to categorical representation happens by applying the trained image encoder to the provided pixel tensor to obtain the hidden vector. CLIP also grounds language in this same representation space by applying the language encoder to the text. Furthermore, even if the model only gives a similarity score between language and images and not an image $\rightarrow$ symbol mapping directly, the latter can be obtained by selecting the word from the dictionary that obtains the highest similarity score. So we see CLIP solves the first part of language grounding as defined by Harnad and part of the second one: one can get words from visual input. The compositional and systematic part of symbol systems is lacking nevertheless, as ranking all possible sentences according to their CLIP score to describe an image is computationally infeasible. One cannot use a sentence or paragraph to describe the image. This is also seen in the lack of exact compositionality exhibited by text-to-image methods based on CLIP.

However today's language models exhibit linguistic ability and behavior that suggests that they do operate at least in part as a capable symbol system, as if the words themselves have meaning since they are used in all the right ways. Since no grounding in sensory information is performed, how is this possible? It looks as if the symbol grounding problem has been solved without even a ground. Looking at a language model's operation more closely, we realize that the symbols manipulated by these models are, as in the case of language in CLIP, grounded in abstract, distributed representations living in the hidden spaces of the model. What happens here is that as training progresses in LLMs, these abstract representations gain a dynamics that reflects very closely the dynamics of perceptual objects, without any actual contact with the perceptual objects themselves! And if these learned

dynamics accurately reflect the real-world, then this LLM can be used to perform actual tasks. This is what (Carta et al., 2023) call *functional grounding* (See Figure **??**).

So if we define grounding as *functional* grounding, we are in the unexpected situation where multimodal models (like CLIP-based methods) are actually less grounded than models trained on text alone, since the dynamics of their internal representations reflect less well the dynamics of the real world. To see how higher functional grounding translates into higher competence in the real world, it is very instructive to look at the SayCan system (Ahn et al., 2022a). This method uses a pretrained language model to guide task completion in real environments, in a robot. For a given high-level goal, the language model provides candidate lower-level instructions and these get ranked conditioned on their values as computed by a goal-conditioned policy that is itself grounded in the robot's context. If these context-grounded values are not used by the language model, it will propose plans of actions that are not contextually relevant. So here we have an intermediate situation: the LLM is *functionally* grounded in its training data (the whole internet), but not in this specific environment. Its knowledge might not match the possibilities at hand. The goal-conditioned value functions are needed to fully align the LLM to its environment. Complete functional grounding can also be achieved by further training on data collected in the current context, as (Carta et al., 2023) demonstrate with RL-finetuned LMs on text-only environments. Here grounding the language model doesn't mean grounding it in the sense of (1), but indeed in the sense of (2): functional grounding.

Grounding is thus fundamental for the symbols used by language to mean anything, but this grounding doesn't necessarily mean connecting the symbols in a low-level perceptual reality: rather the operation of the symbol system must follow rules that are consistent with the rules of the outside world. LLMs have demonstrated that it is possible to approximate this with unsupervised training, and (Carta et al., 2023) show that further grounding can be achived by interacting with an environment, even if purely linguistic and non-perceptual. Furthermore, by using textual, symbolic environments, we do not need to train a network to perform the correspondance to low-level percepts. Since grounding in language environments is both valid and computationally simpler, this motivates us further to use language environments as our testbed.

2.1.4 Background summary

To summarize here our review of language, we will start by concluding that language enables the expression of an infinite amount of possible meanings. Language, as a uniquely human communication system, is compositional: it is both productive and systematic. This is what allows the listener to understand the contents of an utterance heard from the first time. No other animal demonstrates this powerful zero-shot learning. Linguistic theories, whether they are based on generative grammar or its subsequent theories, or whether they are based on some form of construction grammar, attempt to explain this effect by describing language as a complex recombination of simpler elements and structures, hierarchically. What this means is that human language defines an infinite and vast space to explore, and that users of language can invent sentences and propositions describing situations never encountered before and understand what they mean zero-shot.

In a second time, we have seen that language and thought are intimately linked. Language is a scaffold for thought, helping us to acquire our earliest concepts, and bringing already known concepts together to form complex concepts that would be too costly to discover on our own. Whether or not language is the medium for thought, it seems that something really close to language or internal

discourse happens when we perform reasoning and planning. Processes of imagination, linked with episodic recall, seem to also be related to processes close to language.

In light of these previous arguments, we think it is not only justified but also of great scientific and practical interest to study autotelic agents who operate on language, evolving in linguistic worlds and operating with linguistic goals. Because the possibilities of language are infinite, it allows to define worlds whose possibilities are vast (and classical linguistic theories can guide the construction of rich linguistic environments). But even more importantly, the combinatorial properties of language used as a goal space allows an autotelic agent to imagine goals *that do not exist yet*. Language allows us to imagine impossible ideas like Chomsky's famous *colorless green ideas*, but among all the things we could imagine that do not exist, a select few of them could be brought about by the actions of a powerful autotelic agent. This is the promise of language goals: expressing novel situations. Language acquisition additionally imposes priors on what is possible and what is reasonable, allowing an agent to refine the new goals it will generate for itself. These priors also allow the agent to imagine goals that are aligned with the norms of the community from which this language was learned.

But wouldn't linguistic autotelic agents, with language goals, operating on language-only worlds, be too removed from sensory experience to be a worthy subject of study? Our examination of the symbol grounding problem suggests that this is not in fact the case, as long as the agents are *functionally grounded* in an external environment with systematic rules (physics). The only thing we need for grounding in this sense is that the dynamics of the sub-symbolic tokens be aligned with the dynamics of the environment, e.g. , for the agent to build an accurate model of the world. We can thus confidently simplify the grounding problem by building agents that operate in this space directly.

In the following two contributions, we first move to a complex text-world platform called ScienceWorld and study different designs for autotelic agents: we identify the configuration leading to the highest learning and exploration. We then move on to augment linguistic autotelic agents with language models for goal imagination and relabelling, and show how these agents can both explore and invent novel goals, as well as discover and master goals of interest to a human without being told what these goals are.

2.2 Contribution 1: autotelic agents in a linguistic world

2.2.1 Introduction

We are interested in the problem of building and training open-ended autonomous agents, exploring on their own and mastering a wide diversity of tasks once trained. This can be approached within the autotelic reinforcement learning framework (Colas et al., 2022b). An autotelic agent (auto-telos, one's own goals) is an intrinsically-motivated, goal-conditioned agent equipped with a goal-sampler that uses the agent's previous experience to propose goals for learning and exploration. This goal sampler allows the agent to use previously mastered skills as stepping stones to achieve new ones, and to form a self-curriculum for exploration. This developmental framework is general and is linked to goal-exploration processes like Go-explore (Ecoffet et al., 2021) and adversarial goal generation (Florensa et al., 2018; Campero et al., 2021). Autotelic agents have already been shown to efficiently explore sensorimotor spaces (Péré et al., 2018) and to be able to build their own goal curriculum using learning-progress-based task sampling (Colas et al., 2019b).

Recent work has shown the potential of language to drive autotelic learning (Colas et al., 2022a), both due to its compositional structure and its ability to convey cultural knowledge. For example,

post-episode language feedback by a social peer ($\mathcal{SP}$) can be internalized to identify and imagine future relevant goals (Colas et al., 2020a), acting as a cognitive tool (Clark, 2006). It has also been shown to help scaffold the agent's exploration through abstraction (Mu et al., 2022). It can be reused once the agent is trained for instruction-following (Colas et al., 2022a). Exploring in language space directly is akin to learning to plan at a higher, abstract level; the skills learned at this level can be executed by lower-level modules in an embodied environment (Shridhar et al., 2020; Ahn et al., 2022a; Huang et al., 2022a). Additionally, language conveys our morals and values, and be used as a tool to align autotelic agents towards human preferences (Sigaud et al., 2021).

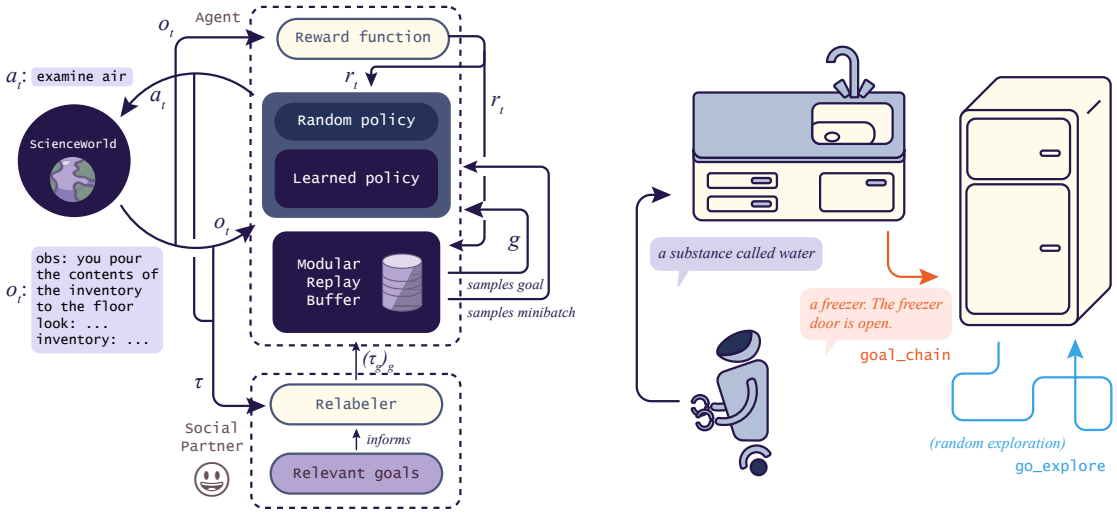


Figure 2.2: Left: overview of the autotelic agent architecture. At the beginning of an episode, a goal g is sampled from the list $\mathcal{G}_a$ maintained in the modular replay buffer. At each timestep t, ScienceWorld emits an observation o_t (see Section 2.2.2). The agent combines o_t and g to decide on an action a_t. o_t and g are also used by the agent to compute the reward r_t. When the episode ends ($r_t \neq 0$ or $t = T$), either the environment is reset, a new goal is sampled or random exploration steps are taken. Right: overview of different exploration methods. The agent is conditioned on a goal and tries to achieve it. In the goal-chain configuration, on achieving a goal or at the end of the allowed T timesteps the agent samples a new goal with a 0.5 probability. In the go-explore configuration, on achieving a goal or at the end of the T timesteps the agent performs 5 timesteps of random exploration.

The common topic in previous work (Colas et al., 2020a; Mirchandani et al., 2021; Mu et al., 2022) on language-based autotelic agents has been to study various forms of exploration mechanisms, some of them relying on language. However, these works did not investigate how to deal with the exploration challenges posed by linguistic spaces themselves, featuring immense action and goal spaces. More precisely, *how specific should $\mathcal{SP}$'s feedback be? How to deal with very hard goals that will be very rarely seen compared to easy ones? How to use easy goals as stepping stones to achieve hard ones?* These are the questions we focus on in this work. For studying this, we place ourselves in ScienceWorld (Wang et al., 2022), a text world (Côté et al., 2019) with very rich dynamics (thermodynamics, biology, electricity) allowing for complex goals such as freezing or boiling water, which requires getting water first, thus defining an optional dependency amongst goals (ScienceWorld is rich enough that goals can be accomplished in multiple ways).

To tackle these challenges, we identify the main drivers of discovery in autotelic agents. In general, there are four ways in which they can discover novel things through goal exploration in an

environment. The agent can **discover from failure**: it can target a goal it misses, and the social peer will relabel this trajectory with the actually achieved goals (if any), an idea exploited in hindsight experience replay (HER) (Andrychowicz et al., 2017). The agent can **discover from babbling**: it is equipped with standard RL exploration mechanisms, such as stochastic policies or epsilon-greedy action sampling. This will induce randomness in goal-conditioned trajectories and allow the agent to stumble onto novel states. The agent can **discover from stepping-stone exploration**: after a goal has been reached or the episode has timed out, the agent can perform random actions or follow another goal from there, a process first investigated by Go-explore methods. Here exploration bonuses (like pseudo-count rewards (Bellemare et al., 2016a)) can be used to make this process more efficient. And finally, the agent can **discover from imagination**: combine known goals to create novel ones, leading to the discovery of novel states and affordances (which can in turn be reused as goals in further exploration). To tackle the exploration challenges posed by large linguistic spaces and nested goals, we especially focus on the first three drivers of discovery. In particular, we show how different methods for learning from further exploration interact together to help the agent navigate the goal hierarchy.

In this work, we study specific challenges posed by autotelic exploration in large language spaces:

1. *How should the $\mathcal{SP}$ provide hindsight feedback (relabeling) to the agent in very large linguistic spaces? Should it be selective or exhaustive?* We show the social peer must give targeted hindsight feedback to the agent to avoid populating the replay buffer with a too wide diversity of detailed goals that prevents making non-trivial discoveries.

2. *In the presence of goals with very different difficulty and occurrences, what is the influence of different goal sampling distributions on the efficiency of learning diverse and complex goals?* We show that the agent needs to bias replay transition sampling towards transitions in trajectories where rare, hard goals are accomplished.

3. *How do methods for learning from stepping-stone exploration influence learning in the goal hierarchy?* We find that sampling goal sequences according to the agent's estimated intermediate competence, and then exploring randomly, significantly improves aggregate competence and reduces variance over seeds.

2.2.2 Problem setting

We are interested in studying the behavior of autonomous agents that freely explore their environments to uncover their possibilities. Such agents are especially needed in environments that are reward-less, or that have sparse or deceptive rewards.

Definitions

Reward-less POMDP Formally, we define a reward-less partially observable Markov decision process (POMDP) (Sutton & Barto, 2018b) as $(S, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O})$, where S is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition function, Ω is the observation space, and $\mathcal{O}$ is the observation function. We also define a trajectory as a sequence of state-action pairs and is represented by $\tau = [(s_0, a_0), \ldots, (s_t, a_t), \ldots, (s_T, a_T)]$ where t is a timestep and T the length of the trajectory.

ScienceWorld Game

> *look*
Kitchen
This room is called the kitchen. In it, you see:
the agent
a substance called air
a counter. On the counter is:
a bowl (containing a red apple, a banana, an orange, a potato),
a freezer. The freezer door is closed.
a fridge. The fridge door is closed.
a glass jar (containing a substance called sodium chloride)
a sink, which is turned off. In the sink is: nothing.
a substance called soap
a **stove**, which is turned off. On the **stove** is: nothing.
a table. On the table is: a **glass cup** (containing nothing).

> *inv*
In your inventory, you see:
an orange.

> *pick up **glass cup***
You move the **glass cup** to the inventory.

> *move **glass cup** to sink*
You move the **glass cup** to the sink.

> *activate sink*
The sink is now activated.

> *look in sink*
Inside the sink is:
a **glass cup** (containing a **substance called water**)
You also see: A drain, which is open.

> *move **glass cup** to **stove***
You move the **glass cup** to the **stove**.

Figure 2.3: A ScienceWorld trajectory illustrating an agent puts a glass cup full of water on the stove. For accessibility purpose, we omit repetitive text (agent receives `obs`, `look`, and `inv` at every step. Italic and boldface denote *action* and **relevant object**.

Autotelic agents In this work, we consider a certain kind of autonomous agents that are driven by an intrinsically-motivated goal-exploration process: autotelic agents. These agents operate without external reward by iteratively targeting goals and trying to reach them, using their own internal goal-achievement (reward) function to measure success. In the process, they observe their own behavior and learn from it. We define $\mathcal{G}_a \subseteq \mathcal{G}$ as the subset of goals experienced by the agent so far during its learning process. We additionally define the agent's goal-conditioned internal reward function as $R : S \times A \times \mathcal{G} \to \{0, 1\}$.

Ideally, we would want to maximize the agent's performance over the entire goal space $\mathcal{G}$, i.e., to find the goal-conditioned policy π that maximizes the expected sum of internal rewards on all goals.

However, these goals are not known in advance and have to be discovered by the agent through structured exploration. This means that this objective cannot be computed and used directly by the agent. Rather, it can be used *a posteriori* by the experimenter as a measure to characterize what the agent discovered and learned.

Social peer Since $\mathcal{G}$ can be quite large, it would be desirable to guide the exploration without forcing it. We consider the agent interacts with a social peer ($\mathcal{SP}$) that gives feedback on the agent's trajectories, i.e., the $\mathcal{SP}$ gives a list of achieved goals at the end of an episode (this list may not be exhaustive and reflects a model of relevance from the perspective of the $\mathcal{SP}$). Formally, we define the social peer as $\mathcal{SP} : (S \times A)^T \rightarrow (\mathcal{G}_{\mathcal{SP}})^m$ which takes in a trajectory and outputs a set of m goals accomplished within this trajectory, where $\mathcal{G}_{\mathcal{SP}} \subseteq \mathcal{G}$ are goals relevant to $\mathcal{SP}$. Then, those accomplished goals can be added to the agent's discovered goals $\mathcal{G}_a$. In principle, goal relabelling can be implemented in many ways, including leveraging pre-trained large language models. In this study, for simplicity, the $\mathcal{SP}$ labels objects presented in trajectories as goals (see Section 2.2.2).

In practice, we are maximizing the agent's performance to achieve the goals it has discovered:

$$\sum_{g \in \mathcal{G}_a} \mathop{\mathbb{E}}_{\tau \sim \pi(.|g)} \Big[\sum_{(s_t, a_t) \in \tau} \gamma^t \, R(s_t, a_t, g) \Big] \tag{2.1}$$

while $\mathcal{G}_a$ converges towards $\mathcal{G}_{\mathcal{SP}}$ as trajectories gets relabelled by $\mathcal{SP}$. In which, γ denotes the discount factor.

ScienceWorld: a text-based environment

ScienceWorld[1] (Wang et al., 2022) is one of the text world frameworks (Jansen, 2022), coming with procedurally-generated household environments and an associated elementary-school science related benchmark. Unlike many other interactive environments that facilitates RL research, in ScienceWorld, the observation space Ω and the action space $\mathcal{A}$ are textual. Specifically, at every timestep, the **observation** consists of three channels (please refer to Figure 2.3 for an in-game example):

• `obs`: a raw observation describing the immediate effect of the agent's actions. We show an example in Figure 2.3, highlighted in green.

• `look`: the description of the agent's surrounding. It is composed of a textual rendering of the underlying object tree, with receptacle-content hierarchy. We show an example in Figure 2.3, highlighted in pink.

• `inv`: a description of the agent's inventory. We show an example in Figure 2.3, highlighted in blue.

The (text-based) **action** space is combinatorial and extremely large. Actions are templated phrases built by composing predicates (from a list of 25 possible unary or binary predicates) with compatible attributes, leading to 200k possible actions at each timestep on average. To alleviate this issue, ScienceWorld provides a shorter list of valid actions $\mathcal{A}_t$ at each timestep t, which makes the action space choice-based. Nevertheless, the size of $\mathcal{A}_t$ is much larger than typical RL environments (on average 800 while in the kitchen, and increases to around 1500 as the agent gets more competent and creates new objects), and these actions are not guaranteed to have an effect to the state, which pose great challenges for RL agents to discover new experiences.

We use elements in room descriptions to represent **goals**. For instance, examples of valid goals for the trajectory shown in Figure 2.3 could be **`the agent`** (which is always true) or **`a substance called sodium chloride`**. This simplified goal representation facilitates relabelling and building the goal-conditioned reward function. As outlined in Section 2.2.1, language-based autotelic

[1] An online demo for ScienceWorld is available at github.com/allenai/ScienceWorld#demo-and-examples

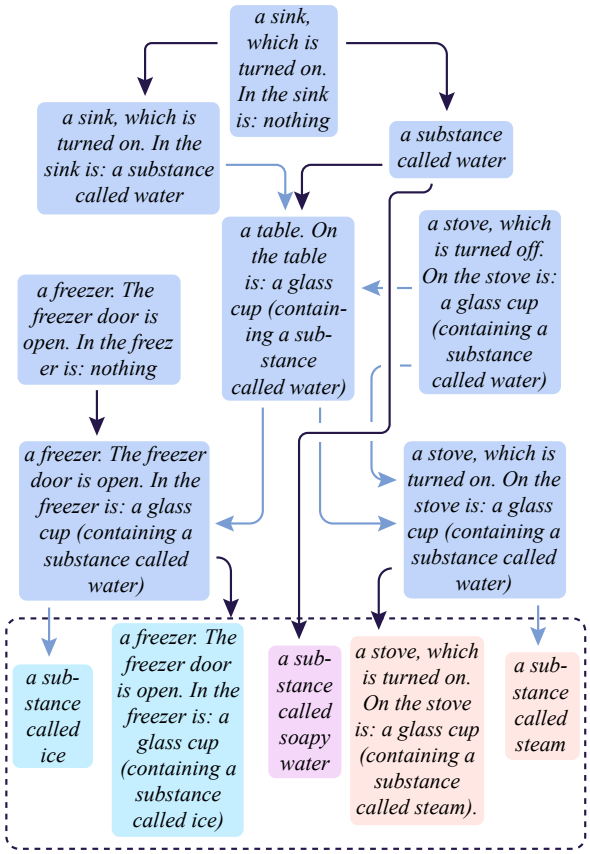


Figure 2.4: Goal hierarchy for goals in $\mathcal{G}_{\mathcal{SP}}$. Goals are object descriptions in the environment (see main text). Light-colored arrows indicate the first goal is helpful for achieving the second one, dark-colored arrows indicate the first goal is necessary to achieve the second one. *Hard goals* are the last goals in the hierarchy, inside the dashed area. For instance, if the goal does not specify steam must be made on the stove, it can be obtained in some other way. The same goes for creating water; if the goal does not specify in which way this must be done, any container will work: either **close drain** once the **sink** is open, or **pour**ing the contents of the **table** (the **glass cup**) into the **sink**.

agents require a reward function to be able to score trajectories against the original goal. Our uni-modal reward function operates in language space, which enables sub-string matching: a goal g is valid if it can be found verbatim in the `look` feedback provided by the environment. While conceptually straightforward, this goal representation is rather expressive. For instance, if the goal targets an immovable object that can only be found in a certain room, this amounts to a navigation goal; if the goal targets an object that is found in a closed receptacle, accomplishing the goal requires opening the receptacle; if the goal targets an object that does not exist in the environment, then the goal amounts to making this object, which can imply a long action sequence.

A song of ice and fire

ScienceWorld tasks are hard exploration problems, as outlined above when considering the number of valid actions per step. In this work, we restrict ourselves to the kitchen to maintain manageable exploration, and we focus on a subset of ScienceWorld tasks: freezing and boiling water. We study the agent's progress through a self-organized curriculum of nested goals of varying

difficulty, guiding the agent towards the most difficult ones. The entire tech tree and its dependency structure is presented in Figure 2.4.

In fact, due to the non-linear design of ScienceWorld, some of the goals can be achieved without following the goal dependency structure: nevertheless this structure is useful to guide the agent's exploration. In this work, we will in particular look at how achieving the first, easier goals can allow the agent to master the harder ones, e.g. how the agent can create its own curriculum for learning. Importantly, this contribution is not about solving the ScienceWorld benchmark itself, but about understanding how multi-goal agents can explore their environment in the presence of large linguistic action spaces and nested goals of very different difficulty.

2.2.3 Autotelic agents in ScienceWorld

Base autotelic agent

Autotelic agents can be implemented with either on-policy or off-policy methods. We adopt the strongest-performing system on the original ScienceWorld benchmark, the deep reinforcement relevance network (DRRN) (He et al., 2016a), as our internal goal-conditioned policy $\pi(.|g)$.

The DRRN is an off-policy deep RL agent equipped with a replay buffer. At inference time, the three observation channels (i.e., `obs`, `look` and `inv`) as well as the goal g are tokenized using a pre-trained sentencepiece [2] tokenizer, they are encoded by a GRU (Cho et al., 2014) and the output representations are concatenated as a vector. In parallel, all valid actions in $\mathcal{A}_t$ are tokenized and encoded with another GRU, to produce a list of valid action encodings. The goal-state encoding is concatenated to all valid action encodings and passed through a 1-layer MLP with ReLU activation to produce $Q(s_t, a)$ for all $a \in \mathcal{A}_t$. The action is either sampled from the resulting distribution (at training time) or the argmax is taken (at evaluation time).

Goal sampling At the beginning of every episode, a linguistic goal g is sampled from $\mathcal{G}_a$, i.e. the set of goals experienced by the agent so far. In the basic case, we assume goal sampling is done uniformly. The agent conditions its policy on that goal towards achieving it. An episode terminates either when the goal is achieved or after T timesteps. Then, the social peer ($\mathcal{SP}$) relabels the trajectory with goals they find relevant, i.e. contained in $\mathcal{G}_{\mathcal{SP}}$, that were effectively achieved during that particular trajectory. The relabelling process is a discrete, linguistic version of hindsight experience replay (HER) (Andrychowicz et al., 2017). The resulting relabelled trajectories, with their associated internal reward, are then pushed into the agent's replay buffer, and any goals discovered in this way are added to $\mathcal{G}_a$.

Goal-modular replay buffer We develop a trajectory-based, modular replay buffer. Specifically, we store each trajectory, paired with accomplished goals (labeled by $\mathcal{SP}$) in an individual slot. During experience replay (in standard deep Q-learning), we use a multi-step strategy to control the transition sampling. First, we sample a goal from the replay buffer using a certain distribution $w(g)$ (uniform unless specified otherwise). Given the goal, we sample a trajectory which has 0.5 probability being a positive example to the goal. If the sampled trajectory is a positive example, we sample a transition from it, with a probability of 0.5 that the transition has a reward.

[2] https://github.com/google/sentencepiece

In preliminary experiments, we observe that the above procedure (especially controlling the amount of reward the agent sees) improves sample efficiency.

Learning To learn the goal-conditioned policy $\pi(.|g)$, we minimize the temporal-difference (TD) loss over transitions in the replay buffer. Given a transition $\tau_t = (s_t, a_t, s_{t+1}, r_t, \mathcal{A}_t, \mathcal{A}_{t+1})$, the TD loss is given by:

$$TD(\tau_t) = l\big(Q(s_t, a_t), (r_t + \gamma \max_{a' \in \mathcal{A}_{t+1}} Q(s_{t+1}, a'))\big), \tag{2.2}$$

where γ is the discount factor and r_t is the internal reward given by $R(s_t, a_t, g)$ when τ_t was first collected. $Q(s_t, a_t)$ is the Q-value for taking action a_t in state s_t and is predicted by the DRRN. The function l is the smooth-L1 loss:

$$l(x, y) = \begin{cases} |x - y| - 0.5 & \text{if } |x - y| > 1; \\ 0.5\,(x - y)^2 & \text{otherwise.} \end{cases} \tag{2.3}$$

To ensure exploration at training time, we add an entropy penalty term which is computed over the Q function with respect to a given s_t. The entropy term H is also normalized by $\log(|\mathcal{A}_t|)$ to account for varying numbers of valid actions across timesteps. Therefore, the final loss is:

$$L(\tau_t) = TD(\tau_t) + H(s_t, (a)_{a \in \mathcal{A}_t}). \tag{2.4}$$

Note there is no separate target network as no particular instability or over-optimism was found in our preliminary experiments.

Discovery from stepping-stone exploration

In this section we present shortly the 2 configurations we use to study the impact of discovery from serendipity in this work: go-explore and goal-chain. Both these mechanisms are explicitly designed to allow the agent to overcome hard-exploration problems and to master nested sets of goals, where achieving the first one is a stepping stone towards mastering the second one. One of the aims of this work is to study this effect on our ScienceWorld goals.

Go-explore This mechanism is very similar to the policy-based version of go-explore (Ecoffet et al., 2021). That is, after sampling a goal g and the policy rollout is terminated (either by completing the goal or completing T timesteps), an additional `num_steps_exploration` actions, set to 5 in what follows, are sampled uniformly from the set of valid actions at each timestep.

Goal-chain This mechanism works in a similar way as `go-explore` but is more deliberate: after goal g is achieved or T timesteps have been achieved, with probability p (0.5 in what follows) another goal g' is sampled and used to condition the policy. Both go-explore and goal-chain can be combined in a single agent.

2.2.4 Experimental results

ScienceWorld is a challenging environment. Baseline agents are barely able to master the simplest tasks of the benchmark (Wang et al., 2022). The large action space and amount of irrelevant state changes the agent can elicit are obstacles to goal discovery, and text environments present challenges for optimization. To answer our questions, we define a set of configurations for agents that we study in what follows:

	Configuration Name	go-explore	goal-chain	eval score (*all*)	eval score (*hard goals*)
♣	**base**	×	×	71.89 ± 16.51	50.52 ± 47.52
	go-explore	✓	×	**80.17** ± 12.37	**59.12** ± 44.47
	chain	×	✓	63.60 ± 19.48	38.24 ± 45.01
	go-explore-chain	✓	✓	77.77 ± 8.81	55.48 ± 43.63
♦	**no-feedback**	×	×	4.18 ± 3.28	0.00 ± 0.00
	unconstrained	×	×	0.00 ± 0.00	0.00 ± 0.00
♠	**uniform-transition**	×	×	50.29 ± 8.80	17.20 ± 36.44
♥	**metacognitive**	✓	✓	**87.31** ± 4.97	**76.36** ± 36.52
	extrinsic-impossible	×	×	67.71 ± 16.18	42.00 ± 45.17

Table 2.1: Main results of our agents on ScienceWorld. We report in the leftmost column the configuration name, in the next two if it uses go-explore or goal-chain. The configurations are clustered by which question they answer. We then report aggregate eval scores on all our goals and aggregate eval scores on our hard goals (defined in Figure 2.4). All eval scores are computed over the last 10 evals for stability, and are averaged across 10 random seeds. See main text for description of the configurations and commentary. NB: **unconstrained** was stopped at 400k timesteps

- **base**: Our baseline agent, as described in section 2.2.3;
- **go-explore**: The **base** agent equipped with go-explore;
- **chain**: The **base** agent equipped with goal-chain;
- **go-explore-chain**: The base agent equipped with go-explore and goal-chain;
- **no-feedback**: A non-autotelic goal-conditioned agent that gets its goals uniformly from $\mathcal{G}_{\mathcal{SP}}$, and without relabeling from $\mathcal{SP}$: it only learns when stumbling upon a targeted goal by normal exploration. It is not using the goal-modular replay buffer (i.e., transition-based instead of trajectory-based).
- **unconstrained**: A **base** agent where the $\mathcal{SP}$ relabels all possible goals ($\mathcal{G}_{\mathcal{SP}} = \mathcal{G}$);
- **uniform-transition**: A base agent, with weights w_g for replay goal-sampling proportional to the total number of transitions for this goal in the replay buffer;
- **metacognitive** An autotelic agent using both go-explore and goal-chain, which samples its goals for an episode according to recorded intermediate competence of these goals;
- **extrinsic-impossible**: A non-autotelic agent that gets its goals uniformly from $\mathcal{G}_{\mathcal{SP}}$, which contains an additional 100 impossible *nonsense* goals.

We train our agent on 800k steps with an episode length of $T = 30$. We use as evaluation metrics the aggregate scores on all our goals as well as evaluation on *hard goals*, that we define as goals where the agent needs to get some water first (see Figure 2.4). Table 2.1 presents the results. We notice the variance is very high in all but the **metacognitive** configuration: this is due to the compounding effects of goal discovery. An agent that stumbles on the easiest goals by chance early in training will be heavily advantaged in its goal-discover compared to an agent that only sees the goal later.

What is the role of $\mathcal{SP}$ relabels in autotelic learning? (♣ vs ♦)

The dynamics of the relabelling process is of paramount importance for any learning to take place at all. In Table 2.1, third section, we present the evaluation scores for a set of experiments with respectively an absent or a talkative $\mathcal{SP}$: in the **no-feedback** configuration the trajectories are simply input with the original goal that was targeted and the associated sparse reward; whereas for

the **unconstrained** configuration, any goal that the agent accomplishes is given by the social peer as a relabel of the current trajectory ($\mathcal{G}_{\mathcal{SP}} == \mathcal{G}$). As in other configurations, goals are given if they are accomplished at any point in time. Since the goal space includes any possible descriptive changes on currently observed objects and that most actions result in such changes, the number of relabelled goals per episode in the **unconstrained** experiments is extraordinary (this agent discovers on the order of 50k goals). Both **no-feedback** and **unconstrained** configurations result in almost-null evaluation scores.

In the **no-feedback** configuration, sparseness of reward is to blame. If we let a random agent explore the room for 800k timesteps, it will encounter goals in $\mathcal{G}_{\mathcal{SP}}$ only a handful of times, and none of the hard ones (see Table 2.2.) For the **unconstrained** configuration, there are such an important number of discovered goals (e.g., containers inside other containers inside other containers, leading to a combinatorial explosion of possible goals). This means that trajectories leading to goals in $G_{\mathcal{SP}}$ are drowned out in the set of other, non-relevant trajectories, and are only sampled a handful of times for replay; the agent's network almost never sees any reward on them, to tell nothing of the optimization process.

Lessons learned: for a multi-task agent to learn correctly in this setting, it needs to have relevant feedback from its social peer, i.e. feedback that is not too descriptive and that is relevant to the goals the social peer wants to instill in the agent.

How to correctly prioritize hard goals in replay for an agent to be able to learn them? (♣ vs ♠)

Another important feature of off-policy multi-goal text agents is their ability to learn a distribution of goals of varying difficulty from their replay buffer. Because the data is collected online, the replay buffer will contain many more exemplars of trajectories for the easy goals compared to the hard goals. With a vanilla replay buffer with uniform replay probabilities over transitions, the transitions corresponding to a difficult goal will hardly get replayed, drowned in the abundance of transitions from easier goals. The ratio of easy-goal transitions to hard-goal transitions only gets worse as the agent achieves mastery of the easy goals and more such transitions fill the buffer. This motivates the use of the modular goal buffer described in section 2.2.3. To empirically validate our choice, we compare the modular buffer with one mimicking the functioning of a basic replay buffer: to do so, instead of sampling goals to replay uniformly, we sample goals with a weight proportional to the number of transitions in all trajectories corresponding to this goal (the **uniform-transition** configuration). This configuration achieves lower performance and plateaus sooner compared to the **base** configuration which serves as our baseline.

Additionally, we investigate whether other replay distribution over goals are important for learning. We investigate difficulty-based sampling (the weight of a goal is given by agent competence over the last 50 attempts of the goal). We also investigate the intermediate difficulty configuration, where the weight given to a goal in goal-sampling w_g depends on empirical competence c on this goal using the following formula:

$$w_g = f_c(c) = \alpha \exp\left(\frac{(c-0.5)^2}{2\sigma^2}\right) + \beta, \tag{2.5}$$

α and β are set to 1.0 and 0.2 respectively. We provide the results in the first row of the Table 2.1. We hardly see any difference between these different goal sampling configurations.

Goal	#occurences
a freezer. The freezer door is open. *In the freezer is: nothing.*	4975
a sink, which is turned on. *In the sink is: nothing.*	4118
a substance called water	473
a sink, which is turned on. *In the sink is: a substance called water.*	68

Table 2.2: Occurrences of goals for a random agent run for 800k timesteps with environment reset every 30 timesteps, similar to the interaction setup of our **base** configuration. We record goals at each timestep as done by $\mathcal{SP}$. All omitted goals have 0 occurrences over the whole random run.

Lessons learned: in a multitask agent with tasks of varying difficulty, where exemplars for these tasks are present at very different rates in the replay buffer, it is important to have a replay mechanism that samples often enough transitions for rare goals.

What is the role of goal distributions when sampling goals for exploration? (♣ vs ♥)

We introduce the **metacognitive** agent configuration (so-called because it uses knowledge of its own competence to target goals): in this configuration the **base** agent is equipped with go-explore, goal-chain and samples its goals based on intermediate competence (as defined in Equation 2.5). The agent performs significantly better than all our other configurations, and also exhibits very low variance (as much as 3 times as low as other agents). The difference is even more apparent if we look at final performance on *hard goals* compared with our **base** configuration (see Figure 2.1, second column). For an autotelic agent, focusing on goals on which it experiences intermediate difficulty allows it to target goals on which there is good learning opportunity, as goals that are too easy are already mastered and benefit less from further exploration, whereas goals that are too hard are still unreachable for the agent.

We finally describe experiments highlighting the interplay of hindsight relabelling with goal sampling. In the **extrinsic-impossible** configuration, the agent is given the list of 14 usual goals of interest plus an additional 100 *nonsense* goals, consisting of the phrase `a substance called` followed by various made-up words. We see that, contrary to intuition, the extrinsic-impossible configuration works rather well: the final evaluation score is similar (different with no significance) from the base configuration. This highlights a very important property of goal-exploration processes: for non-trivial exploration, a very good baseline is having random goal sampling that pushes the agent to have diverse behavior. As long as the agent discovers meaningful states in the environment and the $\mathcal{SP}$'s behavior is helpful, diverse goal-conditioned behavior can be learned; the performance remains lower than in autotelic configurations nevertheless.

We see here that the dynamics of goal sampling are also important in the agent's exploration, and ultimately, learning. An agent that samples goals of intermediate competence creates its own curriculum where goals that are stepping stones are targeted first and further exploration proceeds

from then on. On the other hand, an agent can be given unfeasible goals as long as they lead to diverse enough behavior.

2.2.5 Related work

Autotelic agents, goal-exploration processes, novelty search Autotelic agents were born from the study of intrinsic motivation and curiosity in humans (Oudeyer & Kaplan, 2007a) and the application of these models in developmental robotics at first (Baranes & Oudeyer, 2013b) and machine learning more recently (Forestier et al., 2022a; Colas et al., 2019b). They are very close conceptually to other goal-exploration processes such as Go-explore (Ecoffet et al., 2021). The latter was developed to tackle hard-exploration challenges and stems from insights from novelty search (Lehman & Stanley, 2011a): exploration in environments with sparse or deceptive rewards can be driven by the search for novelty alone. The ability of autotelic agents to self-organize a curriculum (Elman, 1993) of goals for training is a form of automatic curriculum learning (Portelas et al., 2021) and has been studied by adversarial goal generation approaches (Florensa et al., 2018; Campero et al., 2021).

Language-conditioned agents, language for goal-exploration Building language-instructable agents has been one of the aims of AI research since its inception and is still a very active area of research today in machine learning (Anderson et al., 2018a; Luketina et al., 2019) and robotics (Tellex et al., 2020); notable recent breakthroughs were achieved through use of large-scale pre-trained foundation models for planning (Ahn et al., 2022a; Huang et al., 2022a) and multi-modal grounding (Fan et al., 2022; Jiang et al., 2022). Language has been found to be beneficial for goal-exploration as well, by enabling abstraction (Mu et al., 2022; Tam et al., 2022), combination of different abstraction levels (Mirchandani et al., 2021) and goal imagination (Colas et al., 2022a) supported by systematic generalization (Bahdanau et al., 2018). Go-explore has also been studied in the context of text environments (Madotto et al., 2021); albeit in very simple text environments with comparatively few valid actions compared to ScienceWorld and not in a multi-goal setting, as well as having distinct exploration and policy learning phases.

Interactive text environments Text games are of particular importance to research at the intersection of RL and NLP, and thus for the study of language-informed agents. (Côté et al., 2019) introduced TextWorld, the first such text environment, followed by IF environments (Hausknecht et al., 2020). These tasks are notoriously difficult for RL agents due to the large action space and abstract state space. Methods for exploration have been proposed in these contexts such as reducing the action space with LM action generation (Yao et al., 2020) or using novelty bonuses to counter deceptive rewards (Ammanabrolu et al., 2020). These works however did not investigate multigoal contexts (IF games being quite linear in nature) and the necessity to balance tasks of varying difficulty. ScienceWorld (Wang et al., 2022) was explicitly introduced to investigate language model's abilities to act as agents in an interactive environment, but it also features more complexity and openness than other procedural text games and is thus a perfect testbed for language-based autotelic agents.

2.2.6 Discussion

In this work, we have presented a breakdown of the architecture of an autotelic agent, studied on a hard-to-explore part of the ScienceWorld text environment.We have focused on the necessary

internal components for these agents to perform efficient exploration and task learning. In particular, we have highlighted the need for a replay buffer that over-samples rare tasks and a social peer that provides appropriate interaction. This interaction comes in the form of relevant feedback of the agent's behavior but does not necessarily imply, in the case of the social peer directly giving goals to the agent, that the goals can be feasible: only that they lead to interesting interactions with the environment. The agent can shoot for the moon, all that matters is that it goes on to do something interesting and gets relevant feedback. Additionally, letting the agent sample and chain goals of intermediate competence for itself leads increased mastery of the hardest goals in ScienceWorld.

Overall, we are excited by the challenges and opportunities posed by textual autotelic agents. We identify some important directions for future work. First, we only consider one environment variation; distributions of environments could be considered, and generalization could be studied: this can be challenging for current text agents.

Second, more advanced forms of automatic curriculum setting could be implemented, such as ones using learning progress to sample goals (Colas et al., 2019b). Third, goal sampling in this work has been limited to be taken from the list of achieved goals; a truly open-ended autotelic agent should be able to create its own novel goals based on previous achievements. Last but not least, it is worth exploring to integrate a pre-trained large language model (Brown et al., 2020) into various parts of the pipeline, such that the $\mathcal{SP}$ can alleviate the constraints of string-matching, and being able to imagine relevant but unseen goals, by leveraging commonsense knowledge from the language model.

2.3 Contribution 2: LMA3 – Language-model augmented autotelic agents

2.3.1 Introduction

Each human learns an open-ended set of skills across their life: from throwing objects, building Lego structures and drawing stick figures, to perhaps playing tennis professionally, building bridges or conveying emotions through paintings. These skills are not the direct product of evolution but the result of goal-directed learning processes. Although most living creatures pursue goals that directly impact their survival, humans seem to spend most of their time pursuing frivolous goals: e.g. watching movies, playing video games, or taking photographs (Chu & Schulz, 2020).

The field of developmental AI models these evolved tendencies with *intrinsic motivations* (IM), internal reward systems that drive agents to experience interesting situations and explore their environment (Singh et al., 2010; Oudeyer & Kaplan, 2007c). While knowledge-based IMs drive agents to learn about the world (Aubret et al., 2019; Linke et al., 2020), competence-based IMs drive agents to learn to control their environment (Oudeyer & Kaplan, 2007c; Colas et al., 2022b). Agents endowed with these intrinsic motivations are *autotelic*; they are intrinsically driven (*auto*) to learn to represent, generate, pursue and master their own goals (*telos*) (Colas et al., 2022b). Open-ended learning processes require the joint training of a problem generator (e.g. environment dynamics, opponents, goals) and a problem solver: the former challenging the latter in more and more complex scenarios, providing a never-ending curriculum for the problem solver (Schmidhuber, 2013; Wang et al., 2020; Ecoffet et al., 2021; Jiang et al., 2021; Team et al., 2023). Autotelic agents are specifically

[3] ccolas@mit.edu, laetitia.teodorescu@inria.fr

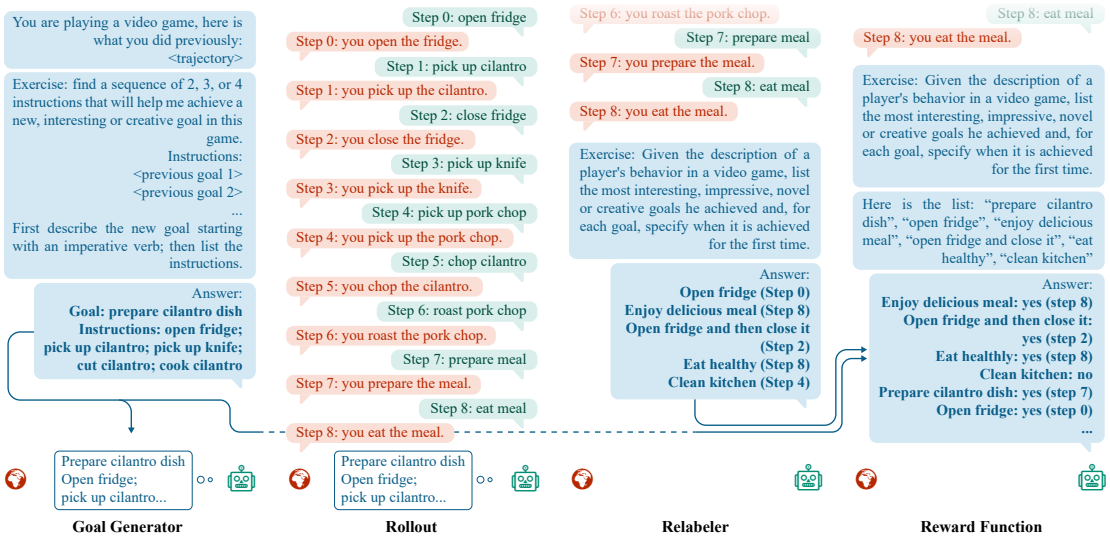


Figure 2.5: **Components of the Language Model Augmented Autotelic Agent (LMA3).** LMA3 agents evolve in a task-agnostic interactive text-based environment. Messages have different color depending on their source: environment is red, LM is blue, learned policy is green. **Goal Generator**: the agent prompts the LM with a previous trajectory and a list of mastered goals to generate a high-level goal and its subgoal decomposition. **Rollout**: The agent then attempts to execute the sequence of subgoals in the environment using its learned policy (green). **Relabeler**: the agent prompts the LM with the trajectory obtained during the rollout and asks for a list of re-descriptions of that trajectory (achieved goals). **Reward Function**: the agent prompts the LM with the trajectory and a list of goals to measure rewards for: the main goal, the subgoals and the goal redescription generated by the relabeler. See complete prompts in Appendix D.1.

designed for open-ended *skill* learning by jointly training a *goal* generator and a *goal-conditioned* policy, see a review in (Colas et al., 2022b).

Human skill learning is a cultural process. Most of the goals we care about are influenced by the goals of others: we clap hands to give encouragements, we strive to finish college, or learn to play the piano. For this reason, we cannot expect autotelic agents learning in isolation to learn to represent goals that we care about, nor to bootstrap an open-ended skill learning process on their own. Just like humans, they should benefit from forms of cultural transmissions; they should learn to pursue other's goals, modify and combine them to build their own, and perhaps influence the goals of others.

Language supports a significant part of cultural transmission; either explicitly via its communication functions: we learn and teach via direct advice, books and online tutorials, or implicitly via its cognitive functions: words help us represent categories (Waxman & Markow, 1995), analogies help us represent abstract knowledge (Gentner, 2016; Dove, 2018), and linguistic productivity helps us generate new ideas by recombining known ones (Chomsky & Lightfoot, 2002). These cultural transmissions let us leverage the skills and knowledge of others across space and time, a phenomenon known as the *cultural ratchet* (Tennie et al., 2009).

Building on these insights, we propose to augment artificial agents with a primitive form of cultural transmission. As current algorithms remain too sample inefficient to interact with humans in real time, we leverage a pretrained language model (LM) as a (crude) model of human interests, biases and common-sense. Our proposed *Language Model Augmented Autotelic Agent* (LMA3) uses an LM to implement: 1) a relabeler that describes the goals achieved in the agent's trajectories, 2) a goal generator that suggests new high-level goals along with their decomposition into subgoals

the agent already masters, and 3) reward functions for each of these goals. We demonstrates the capabilities of this agent in a text-based environment where goals, observations and actions are all textual. Figure 2.5 depicts this process.

The relabeler leverages LMs' ability to segment unstructured sequences into meaningful events (Michelmann et al., 2023). It implements a form of data augmentation (Xiao et al., 2022) that suggests possible future goals to the agent. The goal generator builds on the set of goals already achieved to suggest more abstract goals expressed as sequences of subgoals. It implements a goal-based exploration similar to the one of Go-Explore (Ecoffet et al., 2021). Finally, the reward function ensures that the agent can compute goal-completion signals, the necessary reward signals to implement any kind of goal-directed learning algorithm. Augmented by LMs, our autotelic agents learn large repertoires of skills in a task-agnostic interactive text environment *without any hand-coded goals or goal-specific reward functions.*

The present paper focuses on the generation of diverse, abstract and human-relevant goals in a task-agnostic environment. Because the problem of training a policy that performs and generalizes well to a large set of goals is orthogonal to the tackled issue, we limit ourselves to a crude goal-directed learning algorithm in order to limit the computational budget of calling for an LM API (see computational budget calculations in Section 2.3.5). In Section 2.3.5, we discuss how the insights gained in this contribution can be leveraged to implement more open-ended learning systems.

2.3.2 Related Work

Skill learning can be modeled mathematically as a reinforcement learning problem (RL) (Sutton et al., 1998). In an RL problem, the learning agent perceives the state of the world $s \in \mathcal{S}$, and act on it through actions $a \in \mathcal{A}$. These actions change the world according to a stochastic dynamic function that characterizes the environment $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. An agent learns a skill by training a policy $\Pi : \mathcal{S} \to \mathcal{A}$ to sample action sequences that maximize its expected future returns $\mathcal{R}$ computed from a predefined reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ using a temporal discount factor $\gamma \in [0, 1]$ such that $\mathcal{R} = \sum_t r_t \cdot \gamma^t$. Multi-goal RL problems extend the RL problem to support the learning of multiple skills in parallel (Schaul et al., 2015b). The agent now pursues goals $g \in \mathcal{G}$, each associated with their own reward function R_g and trains a goal-conditioned policy $\Pi^{\mathcal{G}} : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ to learn the corresponding skills. In RL literature, a trajectory τ is a sequence of transitions where each transition is a tuple containing information at a certain time step t: the state s_t and the action taken a_t.

But where do goals come from? Most approaches hand-define the set of admissible goals and corresponding reward functions. They let agents either sample goals uniformly (Schaul et al., 2015b), or build their own curriculum (Portelas et al., 2021). When the goal space is large enough, this can lead to the emergence of diverse and complex skills (Team et al., 2023). Truly open-ended skill learning, however, requires the frequent update of goal representations as a function of the agent's current capabilities — only then can it be never-ending. To this end, the autotelic framework proposes to endow learning agents with an intrinsic motivation to represent and generate their own goals (Colas et al., 2022b).

Learning to represent, imagine and sample goals to learn skills that humans care about requires interactions with human socio-cultural worlds (see argument in introduction, and (Colas et al., 2022a)). Autotelic agents must first internalize the goal representations of humans before they can learn corresponding skills, build upon them and contribute back to a shared human-machine cultural evolution. Goal representations can be learned by inferring reward functions from human

demonstrations (Ng et al., 2000; Arora & Doshi, 2021), via unsupervised representation learning mechanisms (Warde-Farley et al., 2018; Eysenbach et al., 2019; Pong et al., 2020), or by learning to identify the goals achieved in past trajectories from human descriptions (trajectory relabeling, (Andrychowicz et al., 2017; Lynch & Sermanet, 2020; Xiao et al., 2022)).

Building on goal representations learned from linguistic descriptions generated by a simulated social partner, the *Imagine* agent invents new linguistic goals recomposed from known ones (Colas et al., 2020a). Although crude, this goal imagination system allows the agent to pursue and autonomously train on creative goals it imagines, which results in improved systematic generalization and more structured exploration. The present paper extends the *Imagine* approach by leveraging powerful language models to implement several components of the autotelic agent: goal representations, goal-directed reward function and relabeling system. *Imagine* required a (simulated) human in the loop to bootstrap goal representations and could only imagine slight variations of training goals due to its limited imagination algorithm, its lack of grounding and the limited generalization of its reward function. On the other hand, LMA3 does not require any human or engineer input and can generate and master a much wider diversity of goals thanks to the common-sense knowledge and generalization capabilities of LMs.

We evaluate our proposed agent in an interactive text-based environment where observations and actions are all textual (Côté et al., 2019). Text-based environments set aside the challenges of learning from low-level sensors and actuators and focus on higher-level issues: learning in partially observable worlds, learning temporally-extended behaviors, learning the human-like common-sense required to solve these tasks efficiently, etc (He et al., 2016a; Narasimhan et al., 2015; Côté et al., 2019). Text-based environments circumvent the necessity to ground LMs into low-level sensorimotor streams (e.g. visual inputs and low-level motor outputs) and let us focus on the artificial generation of more abstract, human-relevant goals. this contribution is the first to implement autotelic agents with no prior goal representations in text worlds.

Pretrained language models have recently been used to augment RL agents in various ways. In robotics setups, they were used to decompose predefined high-level tasks into sequences of simpler subgoals (Yao et al., 2020; Huang et al., 2022a,b; Ahn et al., 2022b). To limit the hallucination of implausible plans, several extensions further constrain the model by either careful prompting (Singh et al., 2022), by asking the model to generate code-based policies that automatically checks for preconditions before applying actions (Liang et al., 2022a) or by implementing further control models to detect plan failures and prompt the LM to suggest corrected plans (Wang et al., 2023c). LMs can be used to implement a reasoning module to facilitate the resolution of sensorimotor tasks (Dasgupta et al., 2023). They can be finetuned to implement the policy directly (Carta et al., 2023). Closer to our work, the MineDojo approach finetunes a multimodal model to implement a reward function in Minecraft and asks an LM to generate plausible goals to measure the generalization of the reward function (Fan et al., 2022). Finally, the ELLM algorithm prompts an LM to suggest exploratory goals to drive the pretraining of artificial agents in Crafter and HouseKeep (Du et al., 2023b).

Our proposal differs from these approaches in several ways. Our agent is autotelic: it generates its own goals, computes its own rewards. In text-based games, our architecture can handle a large diversity of goals including time-extended ones that can only be evaluated over long trajectories (e.g. *bring the onion to the counter after you've opened and closed the dishwasher*). In contrast, the MineDojo agent has no control over its goals and is limited to generate rewards for a low diversity of short-term goals due to the limited generalization capabilities of the CLIP-based reward (Fan et al., 2022). ELLM generates its own exploration goals but only considers goals that can be reported from a

single state (time-specific goals). It computes rewards using the similarity between the LM-generated goal and descriptions from a captioner, which fundamentally limits the diversity of goals that can be targeted to the list of behaviors the captioner can describe. In the current setup, training or learning a captioner requires some information about the set of behaviors the agent could achieve, which limits the potential for open-ended learning. Compared to ELLM, LMA3 further endows the agent with the ability to perform *hindsight learning* by relabelling past trajectories (Andrychowicz et al., 2017) and the ability to chain subgoals to formulate more complex goals. In contrast with previous approaches (Yao et al., 2020; Huang et al., 2022a,b; Ahn et al., 2022b), we do not leverage expert knowledge to restrict the set of subgoals but learn them online and add the possibility for the agent to use these composed goals as subgoals for future goal compositions.

2.3.3 Methods

This section introduces our learning environment and assumptions (Section 2.3.3), as well as the proposed **Language Model Augmented Autotelic Agent** (LMA3, Section 2.3.3).

Setting and Assumptions

Problem setting. In a task-agnostic environment, we aim to implement the automatic generation of context-sensitive, human-relevant, diverse and creative goal representations. Goal representations are not only goal descriptions but also associated reward functions. Given a sufficiently effective learning algorithm, an autotelic agent endowed with such a goal generation system should learn a large diversity of skills in task-agnostic environments.

Learning environment. We place the LMA3 agent in a text-based environment called *CookingWorld* (Côté et al., 2019; Madotto et al., 2021). The agent receives textual observations and acts via textual commands. Is it not provided with a predefined list of goals or reward functions. The agent is placed in a kitchen filled with furniture (7 including dining chair, fridge, counter, etc), tools (4 including knife, toaster) and ingredients (7 including potatoes, apples, parsley). Across 25 consecutive timesteps, the agent can pick up and put down objects, open and close containers, cut and cook ingredients in various ways, and finally combine them to make recipes. At any step, the agent uses its learned policy to choose an action from the *admissible actions*: the subset ($N \approx 30$-50) of all possible actions ($N = 143$) that the agent can take in the current context (e.g. the agent needs to find and pick up the knife before it can cut any ingredient). Examples of goals the agent could imagine and learn to master include: *slice a yellow potato*, *cook two red ingredients*, *tidy up the kitchen by putting the knife in the cutlery drawer*, *aim to use all three types of potatoes in the dish*, etc.

Assumptions. We make the following assumptions: 1) we only consider text-based environments to allow straightforward compatibility with the LM (see discussion in Section 2.3.5); 2) we assume access to a language model sufficiently large to capture aspects of human common-sense and interests and allow in-context few-shot learning (see implementation aspects below); 3) the agent is spawned in the same deterministic environment at the beginning of each episode. We use a deterministic environment for two reasons. First, because the goal generator needs to know about the environment to generate feasible goals. This is achieved by prompting the goal generator with a past trajectory in that same environment. Second, because it allows us to implement skill learning with a simple evolutionary algorithm, which considerably reduces the sample complexity and thus the cost of querying the LM — albeit to the detriment of generalization. Note, however, that robustness and

generalization of acquired skills are *not* the focus of this contribution, which is interested in the automatic generation of diverse and human-relevant goals. In contrast to most goal-conditioned approaches, we do not assume access to a predefined set of goal representations or reward functions.

Language Model Augmented Autotelic Agent (LMA3)

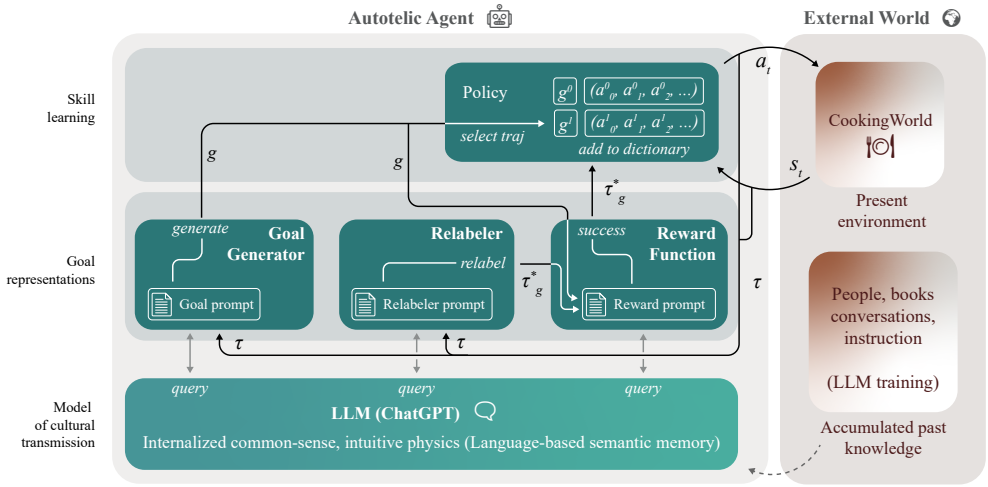


Figure 2.6: **General architecture of LMA3**. LMA3 assumes access to a model of cultural transmission implemented via ChatGPT (dashed line). As shown in the goal representations block, LMA3 leverages that model to generate goals g (left), relabel past trajectories $\tau = (s_0, a_0, ..., s_T, a_T)$ as τ_g^* (middle) and compute rewards when goals are reached (right). s_t and a_t denote the state representation and the action taken at game step t. The goal-conditioned policy (top) attempts to reach its goal within *CookingWorld* (right) and uses relabels and rewards for learning.

General architecture. LMA3 augments a traditional multi-goal learning architecture with goal representations and goal generation powered by an LM (Figure 2.6). In contrast with a standard multi-goal architecture which predefines a bounded distribution of reward functions, the set of goals and associated reward functions supported by the LM is virtually unbounded; it includes all goals expressible with language. The goal generator samples a goal for the goal-conditioned policy to pursue (see **LM Goal Generator** below). Following the hindsight experience replay mechanism (Andrychowicz et al., 2017), we *relabel* trajectories obtained from rolling out the policy and add those to a replay buffer. The relabeling process is implemented by another LM instance which labels up to 10 goals achieved during the trajectory along with their precise completion timestamps (see **LM Relabeler** below). The goal-conditioned policy is then trained with a simple evolutionary algorithm (see **Skill Learning** below). For all prompts, we provide two examples (few-shot prompting (Brown et al., 2020)) and reasoning descriptions (chain-of-thought prompting (Wei et al., 2022a; Kojima et al., 2022)). We provide the complete prompts in Appendix D.1.

LM Goal Generator. In the first episode, the agent does not have any goal representation and simply samples random actions. Then, the agent enters a bootstrapping phase of 4000 episodes for which it uniformly samples goals from the set of discovered goals generated by the relabeler and validated by the reward function (see **LM Relabeler** and **LM Reward Function** below). This allows the agent to first focus on simple goals. After that bootstrapping phase, the agent starts using the LM Goal Generator. At the beginning of each episode, we provide context to the LM by prompting it with the agent's trajectory in the previous episode and a list of up to 60 goals previously reached

by the agent. Then, we ask it to generate a high-level goal and its decomposition in a sequence of 2–4 subgoals from the list. The decomposition lets an agent explore its environment by chaining sub-skills it knows about. This can be seen as an extension of the *Go-Explore* strategy where, after achieving the first goal (*go*), the exploration is further structured towards another goal (*explore*) that is a *plausibly useful* continuation of the first (Ecoffet et al., 2021).

LM Relabeler. After each episode, we prompt the LM with the trajectory of actions and observations and ask for a list of up to 10 descriptions of the goals achieved in the trajectory, as well as specific timestamps for when they were achieved. Note that the goal pursued by the agent in the episode does not matter, the LM Relabeler is free to provide any description of the trajectory. For each goal description we generate a positive trajectory like so: we create a sub-trajectory from step 0 to the step of goal completion, assign a positive reward to the last step and add it to the replay buffer.

We further investigate the impact of leveraging human advice nudging the agent to focus on more abstract and creative goal descriptions. We do so by replacing the 11 simple examples provided in the prompt with 11 more elaborate ones. Instead of describing simple goals involving one action on one specific object (e.g. *roast a white onion*), we provide examples involving sequences of actions (e.g. *use the oven for the second time*), conjunctions of several actions (e.g. *roast an onion and a bell pepper and fry carrots*) or more abstract verbs (e.g. *find out whether the keyholder has something on it*). In contrast with hard-coded relabeling systems, the LM Relabeler uses more linguistic diversity (e.g. synonyms), can describe combinations of actions (e.g. *cook two onions*), or more abstract actions (e.g. *hide an object*).

LM Reward Function. After each episode, we prompt the LM with the trajectory and ask whether the agent achieved any of the following goals: 1) the main high-level goal given by the LM Goal Generator, 2) each of the subgoals (after the bootstrapping phase), 3) each of the relabels generated by the LM Relabeler. 1 and 2 provide feedback to the agent about the goals it was attempting to reach while 3 provides a double check of the redescriptions offered by the LM Relabeler, which could be prone to hallucinations.

Skill Learning. this contribution focuses on the generation of diverse, human-relevant goals and the study of a self-bootstrapped goal imagination and redescription system powered by LMs. To simplify skill learning, we consider a deterministic environment and evolve sequences of actions conditioned on the goal given a simple evolutionary algorithm. For each goal description generated by the LM Relabeler, the agent stores the sequence of actions that led to the completion of that goal in a dictionary. If this goal was achieved previously, it only stores the shortest action sequence. When prompted to achieve a sequence of subgoals, the agent chains the corresponding action sequences together to form the goal-directed policy. Exploration is supported by two mechanisms: 1) by chaining action sequences prompted by the LM Goal Generator and 2) by truncating the action sequence towards the last subgoals in the chain at a uniformly sampled time with probability $\epsilon = 0.2$. After executing the action sequences for all subgoals, and perhaps having truncated the last one, the agent samples actions randomly from the set of admissible actions in proportion of their rarity (i.e. 1 over their occurrence).

2.3.4 Experiments

We compare LMA3 to three ablations and an oracle baseline in the task-agnostic *CookingWorld* environment to assess its learning abilities. For all experiments, we plot the average and standard deviations across 5 seeds. For the different LM modules, we use ChatGPT (*gpt-3.5-turbo-0301*) as

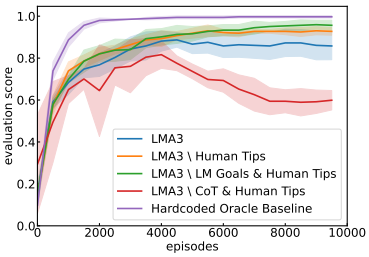
provided through the OpenAI API (Brown et al., 2020). The code will be released publicly with the camera-ready version of the paper.

Ablations and oracle baseline. LMA3 is the first algorithm to allow the automatic generation of linguistic goals and reward functions with no interventions from the engineer. For this reason, there were no obvious baselines to compare LMA3 with. We consider three ablations that remove: 1) the use of human advice in the prompting of the LM Relabeler (*LMA3 \ Human Tips*), 2) the use of human advice and the LM Goal Generator (*LMA3 \ LM Goal & Human Tips*), 3) the use of human advice and chain-of-thought prompting (*LMA3 \ CoT & Human Tips*). In the absence of the LM Goal Generator, goals are uniformly sampled from the set of goals previously discovered, just like in the bootstrapping phase of the LM Goal Generator (see Section 2.3.3). Our baseline is a standard goal-conditioned agent trained on a hard-coded set of 69 goals involving picking up, cooking or cutting objects in the text-based environment (*Hardcoded Oracle Baseline*). This baseline samples goal uniformly from the set of goals previously discovered and uses an oracle relabeler. It implements a standard goal-conditioned policy learning algorithm in text-based environments.

Performance on a human-defined goal space. We want to measure the ability of autotelic agents to learn skills that humans care about. However, autotelic agents learn their own goal representations in worlds that can afford a large space of possible actions and, for this reason, there is no objective set of goals these agents should learn about, no objective evaluation set. We hand-defined a set of 69 evaluation goals involving picking up, cooking or cutting objects in the text-based environment (see list in Appendix A.1). This list is obtained by applying each of the possible action types of the agent (e.g. slice, dice, roast, pick up, put, open, close) to each possible object (e.g. slice+ingredient, open+container) and adding the goal of preparing the recipe the agent can find in the cookbook present in the *CookingWorld*. While the *Hardcoded Oracle Baseline* is explicitly trained on this set of goals and can make use of an oracle reward function and relabeling function, LMA3 variants are not given any prior knowledge of these goals.

Figure 2.7 presents the success rates on this evaluation set computed with the hard-coded reward functions corresponding to each of the 69 goals (not given to LMA3 agents). This metrics evaluates a *minimal requirement*: can LMA3 agents learn to master some of the goals human care about in this world without assuming predefined representations for these goals? Most LMA3 variants learn to reach a large fraction of the evaluation goals, which indicates that leveraging goal representations captured by LMs may support the autonomous learning of skills that humans care about. The *Hardcoded Oracle Baseline* makes use of an *oracle relabeling function* that can faithfully detect any of the 69 goals when it is reached in a trajectory. If we now provide this function to a trained LMA3 agent, we can sweep its memory of action sequences and find the ones achieving the evaluation goals. This form of finetuning does not require further interactions in the environment. Applying it further boosts the success rates of LMA3 agents to near perfect results (see Figure **??**). This shows that LMA3 agents do reach the evaluation goals but sometimes fail to relabel them properly and instead choose to focus on describing other demonstrated behaviors. After finetuning, some of the LMA3 seeds manage to complete the recipe found in the cookbook: a preparation that includes picking up cilantro and parsley from the fridge, opening the cupboard, taking the knife, slicing the parsley, preparing and eating the meal (2 *LMA3* seeds and 3 *LMA3 \ Human Tips* seeds). These results confirm that LMA3 can learn human-relevant goals *completely autonomously*, without relying on any predefined goal representations or reward functions.

Skill diversity. We measure the diversity of discovered skills with 1) the raw number of distinct goals; 2) Hill's numbers, a metric inspired from the evaluation of species diversity in ecosystems



[] []

	Mean ± std
LMA3	0.98 ± 0.020
LMA3 \ Human Tips	0.99 ± 0.007
LMA3 \ LM Goals & Human Tips	0.98 ± 0.000
LMA3 \ CoT & Human Tips	0.96 ± 0.007
Hardcoded Oracle Baseline	1.00 ± 0.005

Figure 2.7: **Performance on human-defined goal space**. Performance on the hand-coded evaluation set containing 69 human-relevant goals, as measured by hard-coded reward functions. (a) across training; (b) at the end of training, after relabeling with the oracle relabeling function and *without further interactions*.

(Jost, 2006) and 3) a metric inspired from the h-index used in research. Hill's numbers allow to make a distinction between species-level diversity (here the type of action required by the goal) and individual-level diversity (e.g. the object on which the action is performed; the object used to perform the action, the location, etc). The type of action required by a goal is inferred as the linguistic stem of the first word of the goal's description. The goal *cutting the apple* is an individual goal of the *cut* goal species. Hill's numbers provide measures of diversity computed as:

$$D^q = \left(\sum_{s \in \text{stems}} p_s^q \right)^{\frac{1}{1-q}},$$

where p_s is the empirical fraction of goals with stem s and q controls the sensitivity of D^q to rare vs. abundant species: D^0 is the count of stems, also called species richness (no sensitivity to species abundance) and D^1 is the exponential of Shannon's entropy, also called perplexity, a measure that quantifies the uncertainty in predicting the stem's identity of a goal uniformly sampled from the set of discovered goals (Jost, 2006). Lower q puts more emphasis on species diversity while higher q puts more emphasis on individual diversity. Note that stems can hide part of the information: different stems might refer to similar behaviors (e.g. *grab* vs *pick up*) while a same stem might refer to different behaviors (e.g. *pick up the apple* vs *pick up the meal*; the 2nd requiring a more complex and time-extended behavior). Such analysis still provides complementary information to the simple goal count. Finally, the stem h-index is computed as the maximum value h such that h stems have at least h goals. This metric is used in research to compute a score mixing diversity and quality of paper citations, here it is used as a way to balance species-level (action type) and individual-level (the rest) diversities.

Figure 2.8 reports the total number of discovered goals, Hill's numbers for $q \in \{0, 1\}$ as well as the stem h-index. While the introduction of the LM Goal Generator (episode 4000) introduces a slight boosts in the diversity of discovered goals (orange vs green), the addition of human advice triggers a lasting increase in goal diversity (blue vs orange). Removing CoT prompting dramatically decreases the diversity of goals discovered by LMA3 (red vs orange). Finally, the diversity of the *Hardcoded Oracle Baseline* remains low as it is restricted to consider the 69 goals from the hand-defined set of goals. LMA3 discovers around 9000 distinct goal redescriptions in 10000 episodes.

Sensitivity to human advice. We measure the sensitivity of LMA3 agents to a small amount of human advice. LMA3 leverages human advice in the prompt of the LM Relabeler, with only a few examples nudging the relabeler to describe more abstract behaviors involving conjunctions

of actions (e.g. *roast an onion and a bell pepper and fry carrots*), repetition of actions (e.g. *open three containers*), non-specific object references (e.g. *slice and cook an orange ingredient*) or more abstract action predicates (e.g. *find out whether the keyholder has something on it*), see full prompts in Appendix D.1. Although the conditions with/without human advice use the same number of examples in their prompts, the more abstract examples used in the full LMA3 condition drives a significant increase in both the diversity of discovered goals (Figure 2.8) and in the abstraction of these goals as measured by the proportion of goals containing conjunctions ("and", "two", "three", or "several times"), or category names instead of specific object names ("ingredients", "items", "container", "somewhere", "fruit", "vegetable", or "tool") see Figure 2.9. Note that the effect on category names, although significant, remains small.

Discovery of unique goals. We measure the ability of each agent to discover *unique goals*; goals that were discovered by this particular seed but were not encountered by any other seed across all algorithm conditions. For each seed, we compute the number of such unique goals, report the ratio of that number over the count of all goals discovered by the agent, and give a measure of the novelty of these unique goals by reporting their average distance to the nearest neighbor in the linguistic embedding space of all goals discovered by all seeds of all conditions. This embedding is computed from a pretrained SentenceBERT model (Reimers & Gurevych, 2019). Figure 2.10 shows that LMA3 agents discover more unique goals, not only in number but also in proportion of the total number of goals they discover (b) and that these goals are more novel in average (c).

Mastery of discovered goals. We measure the skill mastery of agents on several subsets of the goals they discover. For each seed, we compute the success rate of the corresponding agent on three sets of 200 evaluation goals: 1) a uniform sample of all goals discovered by the agent; 2) a uniform sample of the goals only they discovered (unique goals) and 3) a uniform sample of the set of all unique goals from other agents (Figure **??**). We evaluate successes with the LM Reward Function. Note that although the environment is deterministic, mistakes in the LM Relabeler or the LM Reward Function could lead agents to mistakenly classify a goal as reached when it is not. To measure the reliability of this imperfect reward function, we computed its confusion matrix given a set of N=100 trajectories using human labels as ground truth. We estimate the probabilities of both false positives and false negatives to 9% (see confusion matrix in Figure **??**).

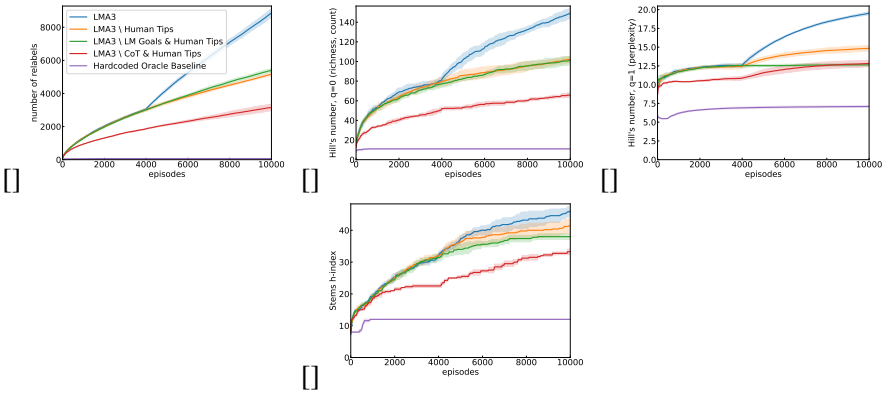


Figure 2.8: **Diversity of achieved goals**: (a) number of relabels, (b) number of stems (Hill's number with q=0), (c) perplexity (Hill's number, p=1), (d) stem's h-index. LMA3 discovers and masters a more diverse set of goals than its ablations. The *Hardcoded Oracle Baseline* is limited to discover goals from the hand-defined set of 69 goals and thus demonstrates very little diversity.

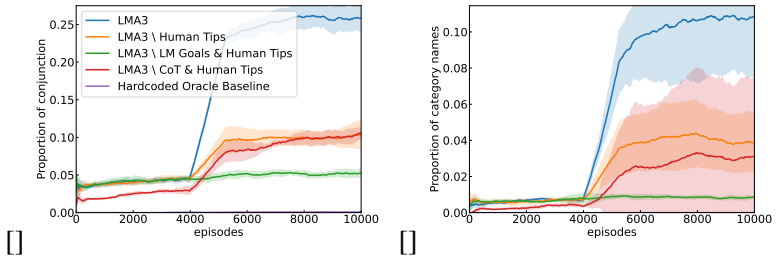


Figure 2.9: **Discovery of more complex and abstract goals**. LMA3 discovers and masters more complex goals expressed as combinations of simpler goals, or using category names (e.g. *ingredients*, *containers*) instead of specific object name as they appear in *CookingWorld* (e.g. *yellow potato*, *kitchen drawer*).

Examples of discovered goals. As discussed above, LMA3 agents discover most of the hand-defined evaluation goals (e.g. *slice a yellow potato*, *pick up the knife*, *open the fridge*). They also learn to consider goals expressed as conjunctions or disjunctions of simpler goals (e.g. *cook two red ingredients*, *put a potato red or yellow in the kitchen cupboard*, *examine an object in the kitchen, like the oven, yellow potato, green or red apple*). They sometimes express goals in more abstract ways (e.g. *wield the knife*, *waste food*, *tidy up the kitchen by putting the knife in the cutlery drawer*, *pack potatoes and apples in the dishwasher*, *refrigerate the yellow apple*). They can use object attributes or object categories to refer to sets of objects (e.g. *put a yellow ingredient in the kitchen cupboard*, *aim to use all three types of potatoes in the dish*, *choose to place an ingredient on a dining chair instead of the counter*). In contrast with hand-defined goals, these goals use new words, category words, or abstract action predicates that do not appear in the vocabulary of the text-based environment.

Finally, the LM Goal Generator generates more complex goals and decomposes them into subgoals the agent masters: e.g. *rearrange the yellow apple and yellow potato inside the kitchen*

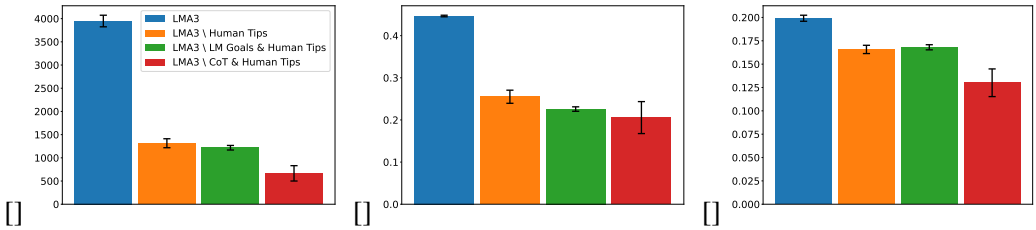
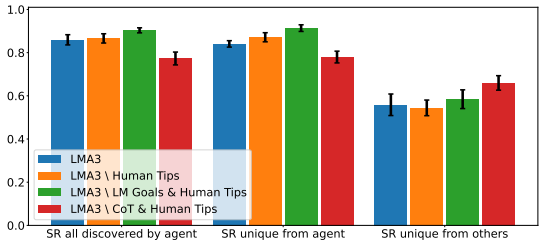


Figure 2.10: **Discovery of unique goals**. For each algorithm, average number of *unique goals* that each agent was the only one to discover (a), ratio of *unique goals* over the count of all goals discovered by the agent (b), and average novelty of the unique goals computed in sentence embedding space (c).



	LM success	LM failure
True success	0.91	0.09
True failure	0.09	0.91

Figure 2.11: **Self-evaluated performance**. (a): Average success rates across seeds for each algorithm when computed on goals discovered during training (left), on unique goals discovered by the agent and no other agent (middle), on a sample of unique goals from other agents, not discovered by the evaluated agent (right). (b): confusion matrix of the LM Reward Function tested over 100 human-relabeled trajectories (56 true success / 44 true failures).

cupboard → [*pick up the yellow apple*, *take the yellow potato*, *place the yellow potato in the kitchen cabinet*, *place the yellow apple in the cupboard*]; *assemble a meal with a fried yellow potato and a roasted red apple on a dining chair* → [*fry the yellow potato*, *cook a red apple in the oven*, *place the red apple and the yellow potato on the dining chair*]; *serve a meal consisting of a roasted sliced yellow potato and a fried diced green apple* → [*dice and fry the green apple*, *slice a yellow potato*, *cook the yellow potato in the oven*]. We found that the LM Goal Generator only generated a low diversity of complex goals: they all prompt the agent to prepare some form of recipe and vary in the properties of the recipes (e.g. *vegetarian*, *colorful*), or its particular ingredients (e.g. *with roasted sliced potatoes*). This behavior stems from the fact that *CookingWorld* is a relatively narrow environment: a kitchen with ingredients and kitchen appliances.

Conceptual comparisons to other skill discovery approaches. Implementing the *Imagine* agent would require the definition of 1) a simulated human describing some of the agent's behavior (e.g. the 69 oracle goals) and 2) a hard-coded symbolic goal imagination system (Colas et al., 2020a). The original imagination system would be limited to imagine linguistic recombinations of the training goals: e.g. from *cut the apple*, *pick the apple* and *cut the parsley*, *Imagine* could generate *pick up the parsley*. These recombinations are either non-semantic (e.g. *open the parsley*) or already contained in the set of 69 goals. As a result, Imagine would be strictly less powerful than the Oracle Baseline: it would not imagine more goals but would need to learn a reward function from descriptions where the Oracle Baseline assumes oracle reward functions.

Other skill discovery methods are limited to low-level goal representations. Visual goal-conditioned approaches learn goal representations by training a variational auto-encoder on experienced visual states and train the agent to imagine and reach new goals in that visual embedding space (Pong et al., 2020). One could imagine a variant of these approaches embedding linguistic trajectories in such a generative model but the resulting skills would not be particularly semantically meaningful. Unsupervised skill discovery approaches co-train a skill discriminator and a skill policy with an empowerment reward (Eysenbach et al., 2019). A variant of these approaches for CookingWorld could consist in co-training a captioner (skill discriminator) and a policy to maximize the likelihood of the trajectory's caption being the original goal of the policy. This algorithm does not exist yet and it is unclear how the agent should sample goals.

2.3.5 Conclusion and Discussion

This section introduced LMA3, an autotelic agent augmented with a language model capturing key aspects of human common-sense and interests to support the generation of diverse, abstract, human-relevant goals. In the *CookingWorld*, LMA3 can learn a large set of skills relevant to humans without relying on any predefined goal representations or reward functions. The diversity of goal representations is further impacted by careful prompting involving chain-of-thought reasoning (Kojima et al., 2022), a small quantity of human-generated advice and the use of an LM-based goal generator.

LMA3 can be applied in any environment where the agent's behavior can be described with language. Although it does not cover all possible scenarios, many of the skills that humans care about can be described with language: from simple actions like picking up a glass, to more abstract behaviors like composing a haiku or coding a sorting algorithm. Lower-level behaviors, on the other hand, are hard to express with language (e.g. fine-grain robotic manipulation). For such behaviors, future work could combine LMA3 with unsupervised skill discovery algorithms such as DIAYN

(Eysenbach et al., 2019) or Skew-Fit (Pong et al., 2020). Modular autotelic agents could then target goals from several goal spaces in parallel and perform cross-modal hindsight learning as proposed in (Colas et al., 2019a).

this contribution focused on goal generation and used a simple skill learning approach to limit the sample complexity of the experiments. Future work could build on the proposed approach to achieve more open-ended skill learning. Let us discuss the key elements that should be improved to that end. A key aspect of open-ended learning is the co-adaptation of the goal generator and the goal-reaching policy. In LMA3, goal generation evolves as the LM Goal Generator recursively composes subgoals the agent masters towards more complex goals. In addition, the LM Relabeler should also adapt and describe harder and harder goals as the agent learns to master them. To this end, the agent should be given the ability to track its own performance (estimated success rate, learning progress or uncertainty) and use these metrics as intrinsic motivations to guide relabeling. Improving the skill learning algorithm would also help LMA3 generalize to a larger diversity of goals and thus focus on harder ones faster. Examples of more sophisticated skill learning algorithms include: leveraging deep reinforcement learning approaches (Hessel et al., 2018) with transformer-based architectures (Chen et al., 2021a; Janner et al., 2021), finetuning another large language model using online interactions (Carta et al., 2023), or leveraging state-of-the-art model-based approaches (Hafner et al., 2023).

Given these more scalable learning approaches, one should consider the exploration of larger worlds. The set of possible interactions in *CookingWorld* is fundamentally limited to a number of distinct interactions with a few objects. As goal generation gets more abstract and diverse and the skill learning approach learns more skills, the main bottleneck becomes the complexity of the environment. While current text-based environments are typically not open worlds, one might consider the use of non-textual open worlds such as Minecraft, coupled with image- or video-to-text captioning systems. To this day, open-source multimodal model cannot relabel trajectories in an open-ended way, and even state-of-the-art closed-source variants still require finetuning before they can be used as success detectors in specific environments (Du et al., 2023a). The multimodal version of GPT-4 may change that in the near future, and could be easily integrated within the LMA3 framework. Recent work Wang et al. (2023a) has also considered Minecraft as a text world through a programming interface, see more discussion in Chapter 4.

Although LMs are a useful resource, they remain expensive not only to train but also to use. The experiments presented in this contribution represent 550k calls to ChatGPT of about 4k tokens per call. This represents ≈ USD 4,400 with the public pricing of USD 0.002 /1k tokens. A given seed of LMA3 run for 10k episodes and costs about USD 240. Training large neural policies that generalize well in complex environments would require about two orders of magnitudes more episodes (≈1M), which would raise the cost of any single seed to ≈ USD 22,400. Pushing these ideas forward may thus require a combination of reduction in inference costs and prices, the distillation of LMs into smaller environment specific reward functions and relabeling functions.

As the field moves towards more and more open-ended agents, their evaluation becomes more complex. How should we evaluate agents that imagine their own goals, specialize in certain skills and not others? this contribution used the diversity of stems as a proxy for the diversity of interaction types the agent could learn to demonstrate. Other metrics could include the measure of diversity in linguistic space, various measures of exploration computed from the agent's behavior, human studies to evaluate the diversity, creativity, abstraction and complexity of the mastered skills. The evaluation of the relevance of learned skills for humans could also require humans in the loop interactively testing the capacities of agent in standardized interaction protocols.

2.4 Chapter conclusion

As we draw towards the end of this chapter, let us recapitulate what we have learned, and how this answers our original questions about linguistic autotelic agents.

In **Contribution 1** we have presented a study of linguistic autotelic agents and we have established that goal sampling in the goal generator as well as in the replay buffer is more effective when guided by intermediate competence, in line with moderate-arousal (Berlyne, 1960) and flow (Csikszentmihalyi, 1990) theories of intrinsic motivation. We have established that in the framework of off-policy deep reinforcement learning with a replay buffer, it is of crucial importance that the Social Partner, providing hindsight feedback on achieved goals after a trajectory, should restricted to labelling interesting and relevant goals. We could call this property hindsight relevance. Our results additionally suggest that it might be useful to explore randomly after a discovered goal as well as to set a new goal once the previous one was achieved, in line with the go-explore strategy. However, in this work we did not use the open-ended properties of language to flexibly imagine new goals, restricting ourselves to a list of nested goals the social partner could relabel.

In **Contribution 2** we have expanded this approach by presenting LMA3, augmenting linguistic autotelic agents with language models. An LLM is used as a goal-imagination module as well as for scoring arbitrary goals and for relabelling them. LLM priors ensure that the invented goals are linked to the environment and well-formed. The LM goal relabeler implicitly includes hindsight relevance. This allows our LMA3 agent to learn a very diverse set of skills, and to ensure that these skills are mostly aligned with what humans consider relevant in a particular environment. While this approach solves the issue of how to imagine complex goals by flexibly recombining simpler ones and taking into acount commonsense knowledge about the environment, there are still obstacles for open-endedness, some technical, some conceptual.

The main technical limitation in the previous contributions is the ineffectiveness of skill learning with reinforcement learning from scratch. In our ScienceWorld study, we observe very high variability across runs and very low generalization, preventing us to scale the approach to larger environments. In preliminary studies done with the LMA3 agent we used a similar method (based on a deep RL algorithm close to the one we used in our ScienceWorld study); however we abandoned it due to its low sample efficiency: mastering any given skill would have taken at least several hundred episodes. This would have meant a prohibitive number of calls to the LLM would have been needed. Low generalization would have been an important obstacle to learning as well. This limitation is an important obstacle to scaling.

However, the conceptual limitations to our contributions are more profound. The first one is the insufficient coupling between goal generation and skill learning. In the ScienceWorld agent, this is achieved by sampling known goals of intermediate competence: however these goals are never novel, since they need to be experienced at least once to be sampled. In LMA3, a small subset of achieved goals is presented to the LLM which composes them to imagine something more complex: but the small number of goals presented and the lack of a metacognitive estimate of the goal difficulty prevents the model from truly scaling up the difficulty of its imagined goals and relabels. How can we implement agents which use their own achievements to challenge themselves with open-ended, creative goals? How can we correctly condition on previously achieved goals

The second conceptual limitation of our contributions of this chapter is the intrinsic limitation of the environments we use. At the beginning of this section we have argued that language can define open-ended environments. While this is true, we have limited ourselves to procedural environments

with simple underlying rules and templated language. While ScienceWorld in particular is quite rich, most of this richness comes from arbitrary object stacking, which, while technically infinite, has a very simple structure: the recursive application of the containment relation. This does not make for a very compelling vision for linguistic open-endedness. How can we scale environments without losing the precise notion of environment dynamics and goal-achievement that environments based on rules and object trees gives us?

Chapter 3

Object-Based Representations and Language Grounding

Preamble to the chapter

This second experimental chapter provides some elements for tackling the language grounding problem (see also Section 2.1.3). What could be good principles for an agent to link its language goals to its sensorimotor observations? After an introduction on grounding, arguments for focusing on the reward function, and object-based architectures such as Graph Neural Networks (GNNs: Kipf & Welling (2016); Battaglia et al. (2018a)) (Section 3.1), we move on to our third contribution (Section 3.2.2). In this study we introduce SpatialSim, an ensemble of tasks of identification and comparison of spatial configurations of objects. We test several neural network architectures based on GNNs as well as powerful computer vision architectures and conclude that on a series of measures the architectures exhibiting the most relational computation perform best. In our third study (Section 3.3), we ask ourselves what architectural design principles could be needed for grounding language with spatial and temporal semantics. We define an artificial language with the power to describe spatial relations, temporal relations, and temporally-extended predicates, and try to learn a correspondence (or reward function) predicting if a given trajectory of an agent corresponds to the the given sentence. We identify that relational architectures with the least amount of prior assumptions perform best. We then conclude this chapter by an overview of what could be needed to ground truly open-ended language goals, as the ones we aim to study in this thesis. The Sections 3.2.2 and 3.3 reuse material from the following contributions:

- **SpatialSim: Recognizing Spatial Configurations of Objects with Graph Neural Networks**; Laetitia Teodorescu, Katja Hofmann, Pierre-Yves Oudeyer; *Frontiers in Artificial Intelligence*; This contribution was the output of an internship testing ideas for learning structured representations for reward functions, the investigation was led by Laetitia.
- **Grounding Spatio-Temporal Language with Transformers**; Tristan Karch, Laetitia Teodorescu, Katja Hofmann, Clément Moulin-Frier, Pierre-Yves Oudeyer; *Neural Information-Processing Systems, 2021*;
  This contribution came of a desire to extend the environment of Colas et al. (2020b) with more complex language and memory. Tristan and Laetitia devised and performed experiments, ran analyses, and wrote the paper.

3.1 Introduction to the chapter

After studying linguistic-only agents in Chapter (**?**), in this chapter we make first steps towards grounding autotelic agents in sensorimotor spaces. In this background section we will motivate the study of grounding through learned reward functions, outline reasonable design principles for these reward functions and present our contributions that focus more precisely on studying how various architectures perform on reward function grounding tasks. Our main research interest will be on *relational* architectures, e.g. neural architectures based on computing relations between objects. We will, in the following background section, introduce what relational computation is and and motivate the relational architectures studied in the contribution sections.

3.1.1 Connecting the autotelic agent to the sensorimotor world

Multimodal grounding

In the previous chapter, we argued that since language defines an infinite space to explore and should support goal imagination and reflection, we were justified in our study of language-only autotelic agents. We surveyed the symbol grounding problem (Section (**?**)) that arises when trying to connect the symbols of language to meaning in the external world and argued that *functional grounding*, i.e. the alignment between the dynamics of the distributed representations underlying the symbols and the dynamics of the external environment, was all that is needed for a symbol system to embody meaning. Thus a language-only system can be grounded in a language-only world, and act upon it. But while such systems can display infinitely complex behaviors in that space (and that behavior be relevant to us), it can never hope to act in a sensorimotor world without grounding to the modality that it is expected to act upon. This problem of multimodal grounding will the topic of our investigation in this chapter.

There are several subproblems to the issue of multimodal grounding. The first is learning a goal-satisfaction or *reward function*: given an arbitrary goal and an arbitrary trajectory, how well does the trajectory satisfy the goal? The second one is the issue of *goal-conditioned policies*: given a particular goal and an observation of the environment, how can I behave in accordance to this goal? And the third subproblem is *description*, the dual of the policy one: given a trajectory, can one recover a true and relevant description of the trajectory?

The multimodal grounding problem assumes, for a given observation space $\mathcal{O}$ (in the broad sense, including the agent's actions) there is a well-defined, preexisting joint distribution between language utterances λ and trajectories $\tau \in \mathcal{O}^n$: $p(\lambda, \tau)$. These are all possible uses of language among existing, competent locutors of the language in the environment in which the agent is embedded; and it is assumed to be fixed, ignoring effects of language evolution, and it is assumed that each utterance corresponds to an observable state of affairs (possibly temporally extended). In this context, the reward function problem can be characterised as learning an energy function recovering the unnormalized probability distribution $E(L\lambda, \tau) \propto p(\lambda, \tau)$. The goal-conditioned policy learning problem is akin to finding a generative model of trajectories given a language goal λ (approximately, since the trajectories are also limited by the transition function of the environment). The description problem is the problem of learning a generative model of language given a trajectory (and it is easier the policy problem since the agent has complete control on the dynamics of its own utterances).

Learned reward functions

Within the scope of this thesis, we always consider the description problem in the context of hindsight relabelling, as opposed as a target problem in itself, for instance to train communicating agents in multi-agent RL domains. For the purposes of training a policy, if one already has access to a trained reward function, one does not necessarily need to have hindsight relabelling: in principle the RL algorithm could find the optimal policy for a goal from the reward signal alone (especially if only intermediate-difficulty goals are sampled so that success it guaranteed at least part of the time). *From a policy learning perspective*, relabelling is primarily a sample-efficiency measure designed to learn from failure; in hard-exploration domains where the reward is dramatically sparse and intermediate goal difficulty is difficult to optimize for, it could be the only way to learn. However, *from a reward-learning perspective*, relabelling might be more fundamental. Learning an energy or similarity function over language-trajectory couples assumes that these couples are available as training samples to the agent. In the developmental AI framework, these samples are available because the agent starts behaving in an environment with preexisting social peers that relabel the agent's behavior with their existing linguistic knowledge (see Section 2.2 for a discussion of the experimental effects of social partner relabelling on policy learning. Following a Vygotsian perspective, it is only after sufficient exposition to exterior relabelling that the agent internalizes this ability as its own. So, as a consequence of the fact, on the one hand, that learning to produce descriptions are not fundamental to grounding linguistic autotelic agents, and that, on the other hand, externally-produced descriptions are fundamental to acquiring data for reward learning, in what follows we consider the description problem solved (by already existing members of the culture), and datasets of co-occurence of language descriptions and trajectories to be readily available to the agent.

Since policy learning is feasible by optimizing the goal-conditioned reward once we have it, we will also set aside the policy learning part in the investigations of this chapter to focus our object of study on multimodal reward learning alone. Our simplified model in this chapter is thus: the agent is exposed to a co-occurrence of goal and observational data during its development through its own behavior and through passive observation, and learns a reward function from this data. It can then use this base learned reward to learn goal-conditioned policies. In a more realistic scenario of course, the reward learning and the policy learning happen simultaneously, like in the Imagine work of Colas et al. (2020a).

Given that we have motivated the study of reward functions learning from datasets of joint occurrences of goal description and behavior, we would like to study specifically the inductive biases and architectures leading to the most robust learning, and in particular to out-of distribution generalisation. This will be the object of the current chapter. Out of distribution generalization is especially relevant to our purposes. Indeed, for an agent to learn behaviors outside of the coverage of its dataset, the reward function must generalize to novel situations that the policy might encounter. As explained in more detail in Chapter (**?**) we focus especially in the second contribution on the notion of systematic generalization: the generalization ability displayed by an ideal systematic system (see Section (**?**) for a discussion of systematicity in language).

3.1.2 Working memory and the common workspace

Systematicity is a property exhibited most prominently by explicit symbol systems; standard deep learning architectures on the other hand use distributed representations. In the prototypical

architecture, the multi-layer perceptron, all inputs to a layer are connected to all its outputs with different weights, and so on until the outputs of the network. This is a very general system with very few architectural priors, that nonetheless has infinite function approximation capacities in the infinite parameter limit (Hornik et al., 1989; Kidger & Lyons, 2020). The goal of the following sections is to introduce and motivate object-based neural network architectures as a good candidate to solve the systematicity problem in reward functions.

Our two models for working systematic generalization are on the one hand artificial symbol systems such as programming languages, rule systems, and the like, and on the other hand human users of language, who can both infer correctly the meaning of newly observed sentences zero-shot and can do it in a infinitely more flexible way than programming languages and old-school rule-based artificial symbol systems. So inspiration from human cognition might be welcome in building systems that generalize better.

If you recall our discussion of section 2.1.2, we outlined the fact, first supported by Daniel Kahneman (Kahneman, 2011), of 2-tiered cognition composed of a system 1 that is fast, intuitive, unconscious, and a system 2 that is slow, deliberate, and conscious. Michael Tomasello (Tomasello, 2021), in his account of the evolution of behavioral organization on the evolutionary line leading to humans, made a similar decomposition of cognitive architectures in mammals. There would seem to be be an operational tier, concerned with processing observations and actions, and an executive tier charged with planning, reflection, inhibition and representation in a common format from disparate sense modalities. In primates and humans this executive tier becomes complemented respectively by reasoning and cultural processes. System 2 cognition would correspond to these super-operational tiers of cognition whose role is to plan and control action of the lower level tiers. System 2 computation seems to be performed in a common representational workspace, sometimes called the global workspace, and linked with working memory: the type of memory that allows us to keep thoughts "in our mind" for the duration of a task and manipulate them, until focus is broken. A classical result of psychology is that working memory is heavily constrained in capacity, originally found to be around seven elements simultaneously (Miller, 1956) with subsequent models disputing the number of chunks (coherent groups of information (Thalmann et al., 2019)), or even their discrete nature (van den Berg et al., 2014). We will call these hypothesised working memory chunks *objects of thought* and defend the idea that neural networks can be built that perform these very operations.

3.1.3 Objects, concepts, and relational inductive biases

The world appears to us as immediately organized into collections of objects, arranged together in scenes or situations. Our thoughts processes themselves can be characterized, as we have seen above, as a small collection of discrete elements. These are the independent entities that are the support for mental manipulation and language, and can be processed separately and in parallel (Kahneman et al., 1992; Pylyshyn, 2007; Green & Quilty-Dunn, 2017). We will refer to them as *objects of thought*, or objects in short. To implement operations on sets of objects, and relations between objects, into neural networks, there is support in the machine learning community advocating for the use of structured models (Lake et al., 2016; Battaglia et al., 2018b), in particular for using models displaying *relational inductive biases* (Battaglia et al., 2018b). Inductive biases are constraints that bias the convergence of function approximators towards a particular kind of function. In deep learning, the most common form of inductive bias is given by the architecture of the model. *Relational inductive biases* refer to architectures supporting the processing of data composed of separate objects, described in the

same space, and their relations. Graph Neural Networks (GNNs) (Gori et al., 2005; Kipf & Welling, 2016; Gilmer et al., 2017), function approximators operating on graph-structured input (that can return graph, set, or scalar output) naturally implement these inductive biases; this is so because the computation they perform is a composition of 1) independent and parameter-shared operations on object representations and 2) of computations taking as input sets of objects and operating of these objects' features, and possibly on the feature of the edge if it exists (edge computation, representing the relational part). Models based on these ideas have achieved substantial progress compared to unstructured methods in recent years on graph-based data or on more general scenes, whether it is by considering the input as sets of objects in a shared space (Qi et al., 2017; Zaheer et al., 2017) or by explicitly considering the relations between objects (Battaglia et al., 2016; Kipf et al., 2019). We use, in the following contribution both GNNs and Transformers (which are a special case of GNN).

3.1.4 Goals of this chapter

To summarize, in this section we aim to uncover good design principles for multimodal reward architectures, with a special focus on systematic generalization Contribution 3. Do models exhibiting relational structure perform better than unstructured models? What is the impact of hierarchy (spatial, temporal) on learning linguistic concepts?

In Contribution 3 we will temporarily abandon language goals and simply consider relational goals in the space of object configurations. We will try to uncover architectural principles in this domain and conclude that the more general Message-Passing GNN performs better than other GNN variants than Convolutional Neural Networks containing massively more parameters.

In Contribution 4 we will define a simple language inside an environment comprising an embodied agent and objects. This language is a temporally-extended description of agent trajectories. In this context, we compare different architectures and show that within the different relational, Transformer-based architectures we test in this case too much structure can actually hurt learning and generalization.

3.2 Contribution 3: The SpatialSim task

3.2.1 Introduction

As argued in the previous section, objects and their relations seem like a natural representation for states of the world, and thus, by extension, for goals a goal-conditioned agent has to reach. This representation lends itself to geometrical, or spatial, reasoning. Spatial reasoning implies processing configurations of objects and their precise relations in space to judge for instance whether a particular structure of blocks is stable or not (Hamrick et al., 2018), a task that requires fine-grained reasoning over the shapes, orientations and positions of the blocks. These kinds of representations can be acquired in an unsupervised manner directly from the observations (Matthey et al., 2019; Locatello et al., 2020; Greff et al., 2020), which makes them appealing in a context where the agent has to abstract away some of the information in its goal (thus it cannot represent its goal as a target image, it is too low-level) but has no access to language information which would allow it to represent its goals in terms of a language instruction. Object representations thus allow for a moderate level of abstraction in which geometrical information is preserved. They additionally open up the possibility for the agent of learning to achieve object-configuration goals irrespective of its particular point of view on the objects (invariance to geometrical similarity), which is highly desirable: a pyramid

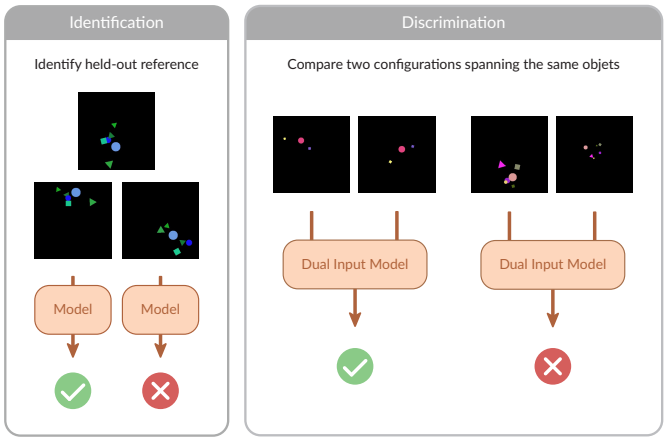


Figure 3.1: Visual illustration of SpatialSim. The benchmark is composed of two tasks, Identification and Discrimination. In Identification, the model is tasked with predicting whether a given configuration is the same as a held-out reference one, up to rotation, translation and scaling. In Discrimination, a dual-input model is tasked with recognizing whether a given pair of configurations is the same up to rotation, translation and scaling. Note: We represent visual renderings of our objects, but note that for all objet-based models we consider the object-based representation is used as input: see main text.

of blocks remains a pyramid regardless of the particular point of view taken by the builder on the scene. A final advantage of representations of goals and states as configurations of objects is their compositional nature (echoing the compositional nature of language): once the agent has learned to stack the blue block on the red block, if it generalizes well it should be able to stack the red block on the blue block.

For all the reasons outlined above, it is thus desirable to make a study of the ability of neural networks to learn to identify and discriminate configurations of objects. However, to our knowledge there is currently no controlled dataset or benchmark allowing to systematically study the problems outlined above. In this work, we introduce SpatialSim (Spatial Similarity), a novel spatial reasoning benchmark, to provide a well-defined and controlled evaluation of the abilities of models to successfully recognize and compare spatial configurations of objects. We divide this into two sub-tasks of increasing complexity: the first is called Identification, and requires to learn to identify a particular arrangement of objects; the second is called Discrimination, and requires to learn to judge whether two presented configurations of objects are the same, up to a change in point of view. Furthermore, we test and analyse the performance of increasingly connected GNNs in this task. We find that GNNs that operate on fully-connected underlying graphs of the objects perform better compared to a less-connected counterpart we call Recurrent Deep Set (RDS), to regular Deep Sets and to unstructured MLPs, suggesting that relational computation between objects is instrumental in solving the SpatialSim tasks. To summarize the key contributions of the paper:

- We introduce and motivate SpatialSim, a set of two spatial reasoning tasks and their associated datasets;
- We compare and analyze the performance of state of the art GNN models on these two tasks. We demonstrate that relational computation is an important component of achieving good performance.

- We provide preliminary analysis in the limits of these models in completely solving the benchmark.

3.2.2 The SpatialSim benchmark

Description

In this work, we consider the problem of learning to recognize whether one spatial configuration of objects is equivalent to another. The notion of equivalence that we consider is grounded in the motivation outlined above: train models that can reason on configurations of objects regardless of their point of view. Because of that, we define equivalence in SpatialSim as geometrical similarity, e.g. any arbitrary composition of translations, rotations, and scalings. In general terms, we frame the problem as a classification problem, where positive examples are drawn from the same similarity equivalence class, and negative examples are drawn from a different one. Since we want, for this first study, to provide the simplest possible version of the problem, we place ourselves in the 2d plane. Extending the setting to 3 dimensions does bring some additional complexity but does not change the underlying mathematical problem (rotations in 2d can be parametrized by their origin and angle, so 3 dimensions, whereas rotations in 3d can be parametrized by their origin and euler angles, so 6 dimensions). We provide a visual illustration of the setting in Figure 3.1.

To provide a clean and controlled benchmark to study the ability of models to learn spatial similarity functions, we work with object features that are already given to the model. We ask the question: supposing we have a perfect feature extractor for objects in a scene, are we able to learn to reliably recognize spatial configurations? Working with objects directly allows us to disentangle the feature extraction process from the spatial reasoning itself.

We define a set of n_{obj} objects as colored shapes in 2d space, each uniquely characterized by a 10-dimensional feature vector. There are three possible shape categories, corresponding to squares, circles and triangles; the shape of each object is encoded as a one-hot vector. The shapes are distributed in continuous 2d space, and their colors belong to the RGB color space, where colors are encoded as a floating-point number between 0 and 1. The objects additionally have a size and an orientation, the latter given in radians. The feature vector for each object is the concatenation of all the previously described features, and a scene containing several objects is given as a set of the individual feature vectors describing each object. Note that the objects are provided without any order, and any permutation of the objects is possible encoding for a given scene; this is to test the permutation-invariance of the models. a highly desirable property.

We subdivide SpatialSim in two tasks. The first one, Identification, allows us to evaluate the abilities of different models to accurately summarize all relevant information to correctly respond to the classification problem. In this task, we sample a random configuration that will be the one the model in required to learn to identify. One configuration corresponds to one datasets, and we evaluate the capacities of the models on a set of configurations. The second task, Discrimination, allows us, in addition to the study allowed by the Identification task, to judge whether the computation learned by a model can be trained to be universal across configurations. For this purpose, the task requires to predict whether two distinct presented configurations belong to the same class or not. We give additional details on data generation for those two settings in the following sections, and a summary table in the annex section A.

First task: Identification

The Identification task is composed of several reference configurations of n_{obj} objects, each corresponding to a distinct dataset. Each sample of one dataset is either in the same similarity equivalence class as the reference configuration, in which case it is a positive example; or in a different similarity class as the reference, in which case it is a negative example. The same objects are present in all samples, not to give the model any additional information unrelated to spatial configurations.

This simple task allows us to isolate how well the model is able to learn as a function of the number of objects n_{obj}: indeed, the model must make a decision that depends on the relationship between each objects, and for this purpose has to aggregate incoming information from n_{obj} vectors: when n_{obj} gets large the information the model is required to summarize increases. This can lead to loss of performance is the capacity of the models stays constant. For this reason, we structure the task with an increasing number of objects: we generate 27 datasets, with $n_{obj} \in [3..30]$. We further group them into 3 collections of low number ($n_{obj} \in [3..8]$), medium number ($n_{obj} \in [9..20]$) and high number ($n_{obj} \in [21..30]$).

For a given number of objects n_{obj} and based on one reference configuration generated by sampling n_{obj} random objects uniformly in 2d space, we generate a balanced dataset composed of:

Positive examples: after applying a small perturbation of factor ε, to each of the objects in the reference, very slightly changing their color, position, size and orientation, we apply a rotation around the configuration barycenter B with an angle $\phi \sim \mathcal{U}([0, 2\pi])$, a scaling,of center B and magnitude $s \sim \mathcal{U}([0.5, 2])$, and a translation of vector t. The latter is sampled from the same distribution as the positions of the different objects;

Negative examples: these examples are generated by applying a small perturbation to the features of the objects in the reference, slightly changing their colors and sizes, and then re-sampling randomly the positions of the objects, while keeping the object's identity (shape, size, color). After randomly resampling the objects' positions, we apply a rotation, scaling and translation drawn from the same distribution as in the positive example to all objects. This is done to ensure no spurious correlations exist that could help models identify positive from negative examples regardless of actual information about the configuration. While it is, in theory, possible to sample a negative example that is close to the positive class, the probability is very low in practice for numbers of objects $n \geq 3$.

For each reference configuration, we generate 10,000 samples for the training dataset, and 5,000 samples, from the same distribution, for the validation and test datasets. For each dataset, we train a model on the train set and test it on the test set. The obtained test accuracies of the models over all datasets are averaged over a group (low, medium and high number of objects).

Second task : Discrimination

In this section, we describe our second task. While in the previous setup the model had to learn to identify a precise configuration, and could learn to perform computation that does not generalize across configurations, we envision Discrimination as a more complex and complete setting where the model has to learn to *compare* two different configurations that are re-drawn for each sample. This task, while being more difficult than Identification, is also more general and more realistic : while sometimes an agent may be confronted with numerous repetitions of the same configuration that it

has to learn to recognize - for instance, humans become, by extensive exposition, quite proficient at the task of recognizing the special configuration of visual elements that is human faces - but a very common task an intelligent agent will be confronted to is entering a new room filled with objects it knows but that are arranged in a novel way, and having to reason on this precise configuration.

For this task, because each sample presents a different set of objects, n_{obj} can vary from one sample to the other, and thus a single dataset can cover a range of number of objects. We generate three distinct datasets, one with $n_{obj} \in [3..8]$, one with $n_{obj} \in [9..20]$ and one with $n_{obj} \in [21..30]$. In preliminary experiments we have observed learning the Discrimination task is very hard, leading to a great dependence on the initialization of networks: some seeds converge to a good accuracy, some don't perform above chance. This is due to the presence of rotations in the allowed transformations in the positive examples; a dataset containing only translations and scalings leads to good learning across initializations. Note that this problem with rotations persists for a simplified setup containing only configurations of $n_{obj} = 3$ objects. To alleviate this and carry the optimization process we introduce a curriculum of five datasets, each one with a different range of allowed rotations in the generation process, with the last one spanning all possible rotation angles.

We generate the dataset as:

Positive examples: we draw the first configuration by randomly sampling the objects' shape, size, color, position and orientation. For obtaining the second configuration, we copy the first one, apply a small perturbation to the features of each object, and apply a random rotation, scaling, and translation to all objects, using the same process as described in the Identification task;

Negative examples: we draw the first configuration as above, apply a small perturbation of magnitude ε and for the second one we randomly re-sample the positions of each object independently, while keeping the other features constant. We finally apply a random rotation, scaling and translation and this gives us our second set of objects.

For each range of number of objects and for each dataset of the curriculum we generate 100,000 samples for the training set. We generate a validation and a testing set of 10,000 samples for each range of n_{obj}. Those datasets contain rotations in the full range.

3.2.3 Models and architectures

With the benchmark, we establish a first set of reference results from existing models in the in the literature, serving to identify their strengths and limits, and as baselines for further work. We consider Message-Passing GNNs for their established performance, notably in physical reasoning tasks, along with stripped-down versions of the same models. Additionally, our hypothesis is that models that implement relational computation between objects will perform best in this benchmark, because it requires taking into account the relative positions between objects and not only their absolute positions in 2d-space. To test this hypothesis, we model a configuration of objects as a graph, where the individual objects are the nodes. We then train three GNN models with decreasing levels of intra-node communication: MPGNN (full Message-Passing GNN, see (Gilmer et al., 2017) performs message-passing updates over the complete graph where all object-object edges are considered; GCN (Kipf & Welling, 2016) computes updated node representations based on the sun-aggregation of linearly transformed node representations from incoming edges, RDS (Recurrent Deep Set) is a Deep Set model (Zaheer et al., 2017) where each object updates its features based on its own features and a global vector aggregating all the other object features (all-to-one message passing); and a regular

Deep Set model where each node updates its own features independently.

We additionally compare, for both tasks, our models to Multi-Layer Perceptron (MLP) baselines. Our MLP baselines are built to have the same order of number of hidden units as the GNNs, to allow for similar representational capacity. However, because there is a considerable amount of weight sharing in GNNs compared to MLPs the number of weights is much higher, and additionally, increases substantially with the number of objects. As a final baseline, and to validate our claims on the importance of relational inductive biases on geometrical reasoning, we compare those architectures operating on object-based input with ResNet18-based architectures operating directly on pixel input.

For all GNN models, after N node updates, the node features are summed, passed through an MLP to output a two-dimensional vector representing the score for the positive and negative class. For the Discrimination task, the models take as input two configurations, pass them through two parallel GNN layers, concatenate their output embeddings and pass this vector through a final MLP to produce scores for both classes. For a detailed description of all models, including a visual overview, please refer to the Appendix Section B.3.

3.2.4 Experimental results

Identification

In this section we report the experimental results for the Identification task. Reported results are averaged over 10 independent runs and across datasets in the same group (low, medium, high number of objects). For a fair Discrimination across parameter numbers in the GNNS, we build each internal MLP used in the message computation, node-wise aggregation, graph-wise aggregation and prediction steps with the same number of hidden layers d and the same number of hidden units h. Since the DS and RDS layer can be seen as stripped-down versions of the MPGNN layer, if d and h stay constant the number of parameters drops for the RDS layer, and drops even further for the DS layer. To allow for a fair Discrimination with roughly the same number of parameters in each model, we use $h = 16$ for all architectures and $d = 1$ for the MPGNN, $d = 2$ for the RDS and $d = 4$ for the DS, and we report the number of parameters in the table. For the embedding vector that aggregates the whole configuration information, we use a dimension $h_u = 16$: thus the number of objects is at first small compared to h_u and gradually becomes larger as n_{obj} increases. We found no significant difference between using one or several rounds of node updating in the GNNs, so all our results were obtained with $N = 1$. We train for 20 epochs with the Adam optimizer (Kingma & Ba, 2014), with a learning rate of 10^{-3} and a batch size of 128.

We report the means and standard deviations of the test accuracies across all independent runs in Table 3.1, as well as the numbers of parameters for each model. Chance is at 0.5. For object-based models, we observe the highest accuracy with the MPGNN model, on all three object ranges considered. It achieves upwards of 0.97 percent accuracy, effectively solving the task for numbers of objects ranging from 3 to 30. Note that in this range performance of the MPGNN stays constant when the number of objects increases. In contrast, the RDS model achieves good performance (0.91) when the number of objects is low, but its performance decreases as the number of objects grows. GCNs perform barely above chance in this task, possibly due to the lower expressivity of linear layers compared to MLPs in the relation operation. Deep Sets show lower performance in all cases, and the MLP achieves 0.85 mean accuracy with $n_{obj} \in [3..8]$ but its performance drops sharply as n_{obj} increases. Finally, we can see that the CNN-based model completely solves the task from pixel

Table 3.1: Test classification accuracies (means and standard deviations are given over datasets and seeds) for the three different models on the Identification task.

Model	$n_{obj} \in [3..8]$	$n_{obj} \in [9..20]$	$n_{obj} \in [21..30]$	Parameters
MPGNN	0.97± 0.026	0.98 ± 0.024	0.98 ± 0.028	2208
GCN	0.54 ± 0.033	0.52 ± 0.014	0.51 ± 0.013	2530
RDS	0.91 ± 0.062	0.85 ± 0.128	0.78 ± 0.19	2038
Deep Set	0.65 ± 0.079	0.60 ± 0.082	0.58 ± 0.09	2386
ResNet18	**0.99** ± 0.017	**1.0** ± 0.007	**1.0** ± 0.013	11.7M
MLP Baseline	0.82 ± 0.09	0.59 ± 0.051	0.56 ± 0.051	6k/48k/139k

input, no matter the number of objects. This suggests that solving this task is possible by simply pattern-recognizing the positive examples with a high-capacity model.

Our results provide evidence for the fact that, while it is possible to identify a given configuration with well-above chance accuracy without performing any relational computation between objects, to effectively solve the task it seems necessary to perform fully-connected message-passing between the objects, especially when the number of objects increases. However, while the number of parameters stays constant in a fully-connected MPGNN when n_{obj} increases, the amount of computation scales as the number of edges ($\mathcal{O}(n_{obj}^2)$). This makes using MPGNNs on complete graphs harder to use at scale. However, note that we consider this task as an abstraction for naturally-occurring configurations of objects that contain a limited number of objects, so this quadratic increase in time complexity should not be a problem in practice. Additionally, we provide evidence that this task is efficiently solvable by large, unstructured architectures.

Discrimination

We now turn to our Discrimination task. We compare the dual-input architecture with different internal layers: MPGNN, GCN, RDS and DS, and to MLP and ResNet18 baselines. The MLP is built according to the principle stated above, and takes as input a concatenation of all the objects in both configurations. As in the previous section, to ensure a fair comparison between models in terms of

Table 3.2: Test classification accuracies for the three different models on the Discrimination task. All metrics were computed on 10 different seeds and trained for 5 epochs on each dataset of the curriculum.

Layer Type	$n_{obj} \in [3..8]$	$n_{obj} \in [9..20]$	$n_{obj} \in [21..30]$	Parameters
MPGNN	**0.89** ± 0.030	**0.81** ± 0.121	**0.71** ± 0.176	4686
GCN	0.55 ± 0.006	0.50 ± 0.004	0.50 ± 0.05	4962
RDS	0.8 ± 0.133	0.68 ± 0.154	0.52 ± 0.04	5326
Deep Set	0.51 ± 0.014	0.50 ± 0.001	0.50 ± 0.005	5274
ResNet18	0.50 ± 0.002	0.50 ± 0.004	0.50 ± 0.005	11.7M
MLP Baseline	0.55 ± 0.002	0.51 ± 0.006	0.50 ± 0.004	26k/192k/552k

the number of parameters we provide our GCN, RDS and DS layers with deeper MLPs. We train models on three datasets respectively with $n_{obj} \in [3..8]$, $n_{obj} \in [9..20]$ and $n_{obj} \in [21..30]$, for 10 seeds per dataset, in Table 3.2. For this task, we train models with the Adam optimizer on 5 epochs on each curriculum dataset (25 epochs total) with a batch size of 128 and a learning rate of 10^{-3}.

We report mean accuracies of the different models and their standard deviation in Table 3.2. As before, chance performance is 0.5. We immediately see the increased difficulty of Discrimination compared to Identification: the model based on MPGNN layers performs best, with mean accuracies of 0.89, 0.81 and 0.71, compared to a performance above 0.97 on Identification. In this case we see the performance drop for MPGNN when n_{obj} increases. RDS performs well above chance when the number of objects is small, but its performance drops rapidly afterwards. GCN performs slightly above chance in the condition with low number of objects, but its performance drops to chance levels afterwards. Both Deeps Sets and the MLP fail to reliably perform above chance. Interestingly, while ResNet18-based models were able to completely solve Identification, they fail to perform above chance in Discrimination, highlighting the harder nature of the task. While ResNet's high capacity allowed it to memorize the positive examples in Identification, the dynamic nature of Discrimination (new configurations are presented at each sample) creates difficulty for unstructured models. As concerns object-based models, we established in the previous section that complete message-passing between nodes is instrumental in learning to identify particular configurations. The experiments in the Discrimination task suggest that layers that allow nodes to have access to information about other nodes are key to achieve good performance, whether this communication is centralized, via conditioning by the graph-level feature as in the RDS, or decentralized, as allowed by the MPGNN layer. Additionnaly, full node-to-node communication seems to be crucial for good performance, and we show in the next subsection how this affects the functions learned by the models. However, it does not seem to be enough to completely solve the task. We provide additional details on the generalization properties of the models in the Appendix.

Model heat maps

In addition to the above quantitative results, we assess the quality of the learned functions for different models. We do this by visualizing the magnitude of the difference between the scores of the positive and negative classes, as output by the different models, as a function of the position of one of the objects in the configuration. We show the results in Figure 3.2. Beyond the clear qualitative differences between different models, this figure clearly shows the shortcomings of the models. A perfect model for this task would show a high-magnitude region in the vicinity of the considered object, and low values everywhere else. Ideally, the object would be placed at a global maximum of this score difference function. Instead, both the RDS and MPGNN models show extended crests of high magnitudes: this means that moving the considered object along those crests would not change the prediction of the models, whereas the configurations are clearly changed. This suggests that all models we considered are limited in their capacity to distinguish classes of similar configurations. This lack of a clearly identifiable global maximum over variations of the position of one object suggests a possible reason for ceiling in performance exhibited by our models: the tested GNNs are unable to break certain symmetries. For the RDS, since each node only has access to global information about an aggregate of the other nodes, it is not surprising to see it exhibit the radial symmetry around the barycenter of the configuration. MPGNNs seem to operate in a different way: for each node the learned function seems to show symmetry around axes related to the principal directions of the distribution of other nodes. The models are thus unable to discriminate a large

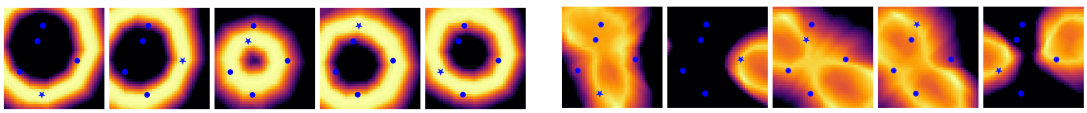


Figure 3.2: Magnitude of the difference in predicted score for the positive and negative classes for a comparison between a 5-object configuration and a perturbed version of this configuration where one object is displaced over the 2d-plane. Value displayed is $score_+ - score_-$, where $score_+$ corresponds to the score of the logits of the positive class as output by the model, and $score$ corresponds to the logits of the negative class. Left row is with RDS layers and right row is with MPGNN. For each row, the displaced object's position is indicated with a blue star, the other ones with a blue dot. The sizes, colors, orientations and shapes of the objects are not represented. Bright yellow means the model assigns the positive class to the configuration where the displaced object would be placed here, black means the negative class would be assigned.

subspace of the configuration space (eg the RDS is insensitive to the rotation of one object around the barycenter of the configuration, since the object stays in the high-magnitude zone). See additional discussion and figures in the Appendix Section B.4.2.

Generalization to different numbers of objects

We have highlighted in the previous section some qualitative measures of the shortcoming of our models in the proposed tasks. To complete these results, in this section we present a generalization experiment for the Discrimination task. Since the models for this task are trained on any couple of configurations, they can be transferred to datasets with higher numbers of objects. In this experiment we train Deep Set, RDS and MPGNN models on one dataset ($n_{obj} \in [3..8]$, $n_{obj} \in [9..20]$ or $n_{obj} \in [21..30]$) and test the models on all three datasets. The results are reported in Table 3.

The results demonstrate the limited abilities of the models to transfer their learned functions to higher or lower numbers of objects. For instance, MPGNNs achieve 0.89 test accuracy when trained *and* tested on 3 to 8 objects, but this performance decreases sharply on the datasets with higher numbers of objects. This is less the case for RDS, presumably because the simpler functions they learn, while achieving lower performance when tested on the matching dataset, are more robust to higher numbers of objects. Another interesting point is that models trained on 9 to 20 numbers of objects appear to transfer better than other conditions. In particular, both RDS and MPGNN achieve higher mean test accuracy when transferring from 9-20 objects to 21-30 objects than models which were directly trained on 21-30 numbers of objects. The 21-30 dataset is harder to train on, so the models trained directly on this dataset may never learn, which bring the mean accuracy down. This suggests that functions useful for good performance on 9-20 numbers of objects are also useful for 21-30 numbers of objects. In contrast, functions useful for good performance on 3-8 numbers of objects do not transfer well to higher numbers of objects.

This evaluation suggests a tradeoff in being able to solve the task well for low numbers of objects versus being able to solve the task for high numbers of objects. This confirms the qualitative evaluation in Section B.4, where we remarked that the functions learned by the models varied greatly with the dataset they were trained on.

	3-8	9-20	21-30
3-8	0.51 ± 0.016 0.80 ± 0.133 **0.89** ± 0.03	0.49 ± 0.046 0.66 ± 0.138 **0.71** ± 0.092	0.50 ± 0.043 0.51 ± 0.048 **0.56** ± 0.075
9-20	0.51 ± 0.046 **0.75** ± 0.125 0.68 ± 0.063	0.50 ± 0.001 0.68 ± 0.154 **0.81** ± 0.121	0.50 ± 0.047 0.52 ± 0.054 **0.68** ± 0.16
21-30	0.50 ± 0.04 **0.60** ± 0.087 0.51 ± 0.048	0.51 ± 0.068 0.68 ± 0.15 **0.77** ± 0.12	0.50 ± 0.05 0.52 ± 0.04 **0.71** ± 0.18

Table 3.3: Generalization results between datasets for Deep Set, RDS and MPGNN. The numbers plotted are averages of testing accuracies. Columns correspond to training datasets, rows to testing datasets. Each block corresponds to one train-set/test-set combination. In each block, the results are given from top to bottom for Deep Set, RDS and MPGNN. Diagonal blocks correspond to matching train set/test set combinations. All reported results are averages and standard deviations over 10 different runs. Rows and columns are annotated with the n_{obj} range.

3.2.5 Related Work

Spatial relations and language

While previous work in Visual Question Answering (VQA (Antol et al., 2015) and Instruction-Following Reinforcement Learning (Luketina et al., 2019) have considered issues related to the ones we consider in SpatialSim, such work is constrained to using spatial relations that can be labelled by natural language. For instance, one of the standard benchmarks in VQA is the CLEVR dataset (Johnson et al., 2016). In CLEVR, questions are asked about an image containing a collection of shapes. The questions themselves are more designed for their compositionality ("the object that is of the same color as the big ball that is left of ...") than for geometric reasoning *per se*. While the dataset does contain questions related to spatial reasoning, there are very few of them (four) and they are of a different nature than the precise reasoning on configurations that we wish to address. For instance, the concepts of "left of" or "right of", while defining broad spatial relations, are not invariant to the point of view of the observer. Since language abstracts away geometry, such a benchmark is unsuitable to the study of geometrical reasoning. Consider the task of stacking rocks of different shapes to make a tower; a language description could be “stack this rock on top of this one which is on top of this one etc...”. But the “on top of” does not capture the precise positional information that allows a rock-stacking agent to estimate the centers of mass to correctly balance its construction. Their linguistic nature makes those datasets unsuitable for investigating the question of learning to identify and compare precise geometrical arrangements of objects. A follow-up to CLEVR is the CLEVRER dataset and tasks (Yi et al., 2019). While this benchmark is very thorough, requiring solutions to be able to perform prediction as well as counterfactual reasoning, we would argue that the tested abilities are a bit broad. In this work, we focus on a more restrained task that allows us to test the abilities of our models to perform recognition of equivalence classes of objects.

Graph Neural Networks

This work is based on the recent line of work on neural networks that operate on graph-structured input. Seminal work (Gori et al., 2005; Scarselli et al., 2009) involved updating the representations of nodes using a contraction map of their local neighborhood until convergence, a computationally expensive process. Follow-up work alleviated this, by proposing neural network models where each layer performs an update of node features based on the previous features and the graph's adjacency matrix, and several such layers are stacked to produce the final output. Notably, Graph Convolutional Networks (GCNs) (Kipf & Welling, 2016; Defferrard et al., 2016; Bruna et al., 2014), in their layers, use a linear, first-order approximation of spectral graph convolution and proved effective in several domains (Duvenaud et al., 2015). However, these works have focused on working on large graphs where the prediction at hand depends on its connectivity structure, and the GCN can learn to encode the structure of their k-hop neighborhoods (for k computation steps). In our case there are a lot less objects, and the precise connectivity is irrelevant since we consider GNNs on fully-connected graphs.

A different class of networks (MPGNNs) has been proposed to more explicitly model an algorithm of message-passing between the nodes of a graph, using the edges of the graph as the underlying structure on which to propagate information. This is the class of model that we consider in this work, because of their flexibility and generality. A variant of this was first proposed to model objects and their interactions (Battaglia et al., 2016; Santoro et al., 2017a) by explicitly performing neural computation on pairs of objects. The full message-passing framework was introduced in (Gilmer et al., 2017) for supervised learning on a molecular dataset, and was expanded in (Battaglia et al., 2018b), which provided a unifying view of existing approaches.

A line of theoretical work has focused on giving guarantees on the representational power of GNNs. (Xu et al., 2018) introduced a simple model that is provably more powerful than GCNs, and universality of approximation theorems for permutation invariant (Maron et al., 2019) and permutation equivariant (Keriven & Peyré, 2019) functions of nodes of a graph have also derived recently. However, the models constructed in these proofs are theoretical and cannot be tractably implemented, so this line of work opens interesting avenues for empirical research in the capacities of GNNs in different practical settings. In particular, (Xu et al., 2018) found that GNNs have powerful theoretical properties if the message-aggregation function is injective. This means that the aggregation function has to preserve all the information from its input, which is a collection (of a priori unbounded size) of vectors, to its output, which is a single vector of bounded dimension. Our setting allows us to examine the ability of GNNs to appropriately aggregate information in an experimental setting.

Finally, our work relates to recent work on learning distance functions over pairs of graphs (Ma et al., 2019). Recent work (Li et al., 2019), in a model they call Graph Matching Network, has proposed using a cross-graph node-to-node attention approach for solving the problem, and compared it to an approach without cross-graph attention, that is closely related to our models for the Discrimination task. However, our work here is distinct, not so much in the actual architectures used, but in the properties of the setting. Their work considered a task related to graph isomorphism, where the precise connectivity of the graph is important. In this work we are interested in the features of the nodes themselves and we consider the underlying graph simply as a structure supporting the GNN computation.

Learning same-difference relations in convolutional neural networks

Related to our work in a recent line of contributions looking to examine the relational reasoning capacities of machine vision systems, in particular of Deep Convolutional Neural Networks (DCNNs) (see (Ricci et al., 2021) for a recent review). This line of work focuses on tasks where a network has to classify images containing two similar or different shapes, usually the two similar shapes being translated and scaled versions of each other. This task, often studied in humans (since (Shepard & Metzler, 1971) in the case of mental rotations), entails abstract relational visual capabilities and it has been shown to be solved by non-human primates (Donderi & Zelnicker, 1969), birds (Katz & Wright, 2006), rodents (Wasserman et al., 2012) and insects (Giurfa et al., 2001). (Kim et al., 2018) have shown that regular DCNNs and Relation Networks (Santoro et al., 2017a) have difficulties learning the same-different abstract categories, a problem that is alleviated by using a Siamese architecture taking as input each of the shapes, a process the authors link to perceptual grouping of objects in humans. Relatedly, (Puebla & Bowers, 2021) have shown that DCNNs evaluated on the same-different task are not robust to changes in the low-level features of the underlying images, arguing that representations of objects and their relations are needed. This is the setting in which we place ourselves in our second task, Discrimination, where object representations are directly accessible to networks with varying amounts of relational computation. In addition, we consider rotational invariance as part of our equivalence classes, which has not been considered in this line of work.

Unsupervised object discovery across time

In recent years there also has been a trend towards using object-centric approaches to physical reasoning, by learning unsupervised latent representations of objects from videos of dynamic object interaction. Constrastive Structured World Models (C-SWMs) (Kipf et al., 2019) learn, from pixel input, a latent representation in as a set of entities, after having passed an encoder; the latents are trained in a contrastive fashion, with positive pairs being continuous in time and negative pairs being randomly matched. This contrastive training scheme ensures the latents capture meaningful information without having to compute any loss in pixel space. AlignNet and OAT (Creswell et al., 2020, 2021) also operate on raw videos of dynamic objects, using a MONet encoder (Matthey et al., 2019) to produce a latent representation of a set of objects. The goal of this work is to find, for any given time step, a permutation between object indices such as the objects resulting from the encoding are always enumerated in the same order. As the focus of the work is not object interaction, their dynamics model is not relational, and is based on an LSTM acting on entity features. The OP3 model (Veerapaneni et al., 2020) builds upon a similar idea than IODINE (Greff et al., 2020) - that is, inferring the entities present in a scene through iterative refinement - extended to temporally-extended image inputs; as such, it also includes a dynamics model based on a GNN. The authors show that using their approach in model-based planning leads to better generalization compared to using unstructured or non-relational approaches. These models have in common that the entity encoder is separate from the dynamics model which operates over a structured latent space. While physical reasoning is feasible using non-relational models (Eslami et al., 2018), each of these works provide evidence for the usefulness of object-centric inductive biases as well as of the separation between object perception models and physics models. SpatialSim focuses on the object-relation component, but this previous line of work suggests how well-performing object encoders could be extracted, even with a very simple encoding scheme (as in the case in C-SWMs), from the spatial and temporal structure of observations of interacting objects. However, the implicit spatial reasoning tasks that are solved in

these tasks are somewhat easier than what is needed in the SpatialSim benchmark because the worlds they consider have smooth transitions: contiguous frames are close to each other in pixel and entity space, which is not the case in our setup.

3.2.6 Conclusion

In this work, we motivated the use of GNNs to learn goal-achievement functions over equivalences classes of spatial configurations of objects. We introduced SpatialSim, a simplified but challenging spatial reasoning benchmark that serves as a first step towards more general geometrical reasoning where a model has to learn to recognize an arrangement of objects irrespective of its point of view. We demonstrated that the relational inductive biases exhibited by Message-Passing GNNs is crucial in achieving good performance on the task, compared to a centralized message-passing scheme or to independent updating of the object features. MPGNNs achieve near-perfect performance on the Identification task, but achieve much lower performance on the Discrimination task. Our experiments with ResNets suggest that object-centered architectures are also instrumental in solving the difficult Discrimination task, as demonstrated by the failure of CNNs with 4 order of magnitude more parameters than the small relational neural networks we consider. Our analysis suggests two shortcomings of current models on this benchmark: 1) the models struggle to accurately summarize information when the ratio between the number of objects and the size of the embedding used for representing the whole configuration becomes large; and 2) GNNs in spite of their relational inductive biases struggle to break certain symmetries; we take this to mean additional theoretical and experimental research is needed to find more appropriate biases for geometrical reasoning.

Now that we have reached these conclusions, the time has come to move to learning the link between an object-structured external world and the language describing it. Do the same general principles hold in a more realistic multimodal domain, a domain that includes not only spatial relations of objects but also predicates and actions described with the past tense?

3.3 Contribution 4: Grounding Spatio-Temporal Language with Transformers

3.3.1 Introduction

In this section we thus turn to the study of *Embodied Language Grounding*. Embodied Language Grounding (Zwaan & Madden, 2005) is the field that studies how agents can align language with their behaviors in order to extract the meaning of linguistic constructions. Early approaches in developmental robotics studied how various machine learning techniques, ranging from neural networks (Sugita & Tani, 2005; Tuci et al., 2011; Hinaut et al., 2014) to non-negative matrix factorization (Mangin et al., 2015), could enable the acquisition of grounded compositional language (Taniguchi et al., 2016; Tani, 2016). This line of work was recently extended using techniques for *Language conditioned Deep Reinforcement Learning* (Luketina et al., 2019). As argued in the introduction of this chapter, among these works we can distinguish mainly three language grounding strategies. The first one consists of directly grounding language in the behavior of agents by training goal-conditioned policies satisfying linguistic instructions (Sugita & Tani, 2005; Tuci et al., 2011; Hill et al., 2020; Hermann et al., 2017; Chaplot et al., 2018). The second aims at extracting the meaning of sentences from mental simulations (i.e. generative models) of possible sensorimotor configurations matching

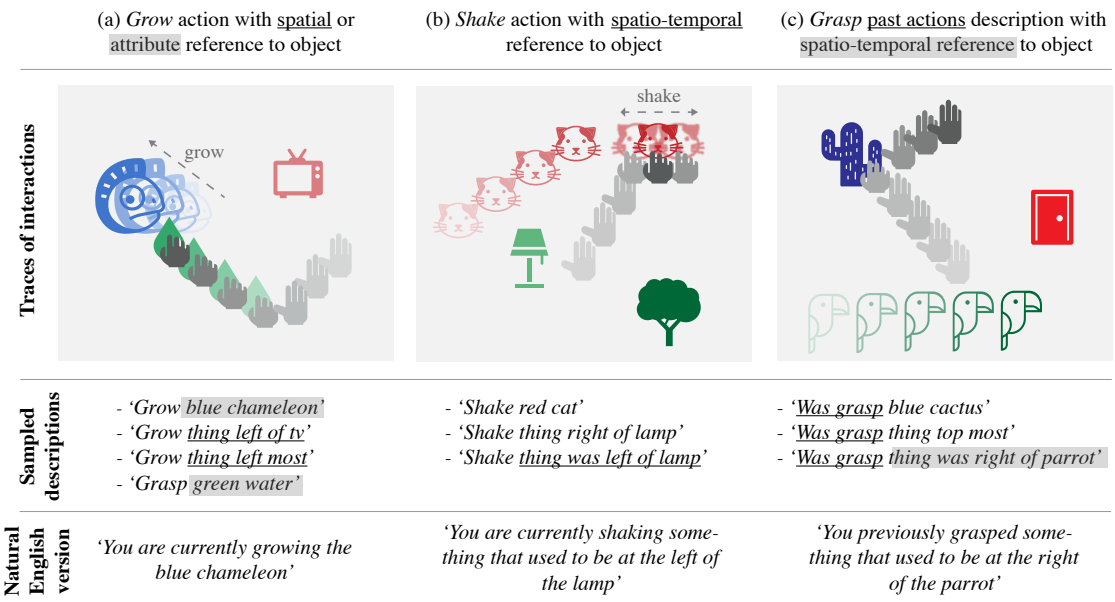


Figure 3.3: **Visual summary of the Temporal Playground environment:** At each episode (column a, b and c), the actions of an agent (represented by a hand) unfold in the environment and generate a trace of interactions between objects and the agent body. Given such a trace, the environment automatically generates a set of synthetic linguistic descriptions that are true at the end of the trace. In (a) the agent grows an object which is described with spatial (underlined) or attribute (highlighted) reference. In (b) it shakes an object which is described with attribute, spatial or spatio-temporal (underlined) reference. In (c) it has grasped an object (past action underlined) which is described with attribute, spatial or spatio-temporal (highlighted) reference. The natural english version is given for illustrative purposes.

linguistic descriptions (Mangin et al., 2015; Akakzia et al., 2021; Cideron et al., 2020; Nguyen et al., 2021). The third strategy searches to learn the meaning of linguistic constructs in terms of outcomes that agents can observe in the environment. This is achieved by training a truth function that detects if descriptions provided by an expert match certain world configurations. This truth function can be obtained via *Inverse Reinforcement Learning* (Zhou & Small, 2020; Bahdanau et al., 2019) or by training a multi-modal binary classifier (Colas et al., 2020a). Previous work (Colas et al., 2020a; Bahdanau et al., 2019) has shown that access to this reward is enough for successfully grounding language in instruction-following agents.

While all the above-mentioned approaches consider language that describes immediate and instantaneous actions, we argue that it is also important for agents to grasp language expressing concepts that are relational and that span multiple time steps. We thus propose to study the grounding of new spatio-temporal concepts enabling agents to ground time extended predicates (Fig. 3.3a) with complex spatio-temporal references to objects (Fig. 3.3b) and understand both present and past tenses (Fig. 3.3c). To do so we choose the third strategy mentioned above, i.e. to train a truth function that predicts when descriptions match traces of experience. This choice is motivated by two important considerations. First, prior work showed that learning truth functions was key to foster generalization (Bahdanau et al., 2019), enabling agents to efficiently self-train policies via goal imagination (Colas et al., 2020a) and goal relabeling (Cideron et al., 2020). Hence the truth function is an important and self-contained component of larger learning systems. Second, this strategy allows to carefully control the distribution of experiences and descriptions perceived by the agent.

We understand the meaning of words by analyzing the relations they state in the world (Gentner

& Loewenstein, 2002). The relationality of spatial and temporal concepts has long been identified in the field of pragmatics (Tenbrink, 2008, 2011) (see Supplementary Section C.3 for additional discussion). Furthermore actions themselves are relations between subjects and objects, and can be defined in terms of affordances of the agent (Gibson, 1968). We acknowledge this and provide the right relational inductive bias (Battaglia et al., 2018a) to our architectures through the use of Transformers (Vaswani et al., 2017). We propose a formalism unifying three variants of a multi-modal transformer inspired by Ding et al. (2020) that implement different relational operations through the use of hierarchical attention. We measure the generalization capabilities of these architectures along three axes 1) generalization to new traces of experience; 2) generalization to randomly held out sentences; 3) generalization to grammar primitives, systematically held out from the training set as in Ruis et al. (2020). We observe that maintaining object identity in the attention computation of our Transformers is instrumental to achieving good performance on generalization overall. We also identify specific relational operations that are key to generalize on certain grammar primitives.

Contributions.this contribution introduces:

1. A new Embodied Language Grounding task focusing on spatio-temporal language;
2. A formalism unifying different relational architectures based on Transformers expressed as a function of mapping and aggregation operations;
3. A systematic study of the generalization capabilities of these architectures and the identification of key components for their success on this task.

3.3.2 Methods

Problem Definition

We consider the setting of an embodied agent behaving in an environment. This agent interacts with the surrounding objects over time, during an episode of fixed length (T). Once this episode is over, an oracle provides exhaustive feedback in a synthetic language about everything that has happened. This language describes actions of the agent over the objects and includes spatial and temporal concepts. The spatial concepts are reference to an object through its spatial relation with others (Fig. 3.3a), and the temporal concepts are the past modality for the actions of the agent (Fig. 3.3c), past modality for spatial relations (Fig. 3.3b), and actions that unfold over time intervals. The histories of states of the agent's body and of the objects over the episode as well as the associated sentences are recorded in a buffer $\mathcal{B}$. From this setting, and echoing previous work on training agents from descriptions, we frame the Embodied Language Grounding problem as learning a parametrized truth function R_θ over couples of observations traces and sentences, tasked with predicting whether a given sentence W is true of a given episode history S or not. Formally, we aim to minimize:

$$\mathbb{E}_{(S,W)\sim\mathcal{B}}\big[\mathcal{L}(R_\theta(S,W), r(S,W))\big]$$

where $\mathcal{L}$ denotes the cross-entropy loss and r denotes the ground truth boolean value for sentence W about trace S.

Temporal Playground

In the absence of any dedicated dataset providing spatio-temporal descriptions from behavioral traces of an agent, we introduce *Temporal Playground* (Fig. 3.3 an environment coupled with a templated grammar designed to study spatio-temporal language grounding. The environment is a 2D world, with procedurally-generated scene containing $N = 3$ objects sampled from 32 different object types belonging to 5 categories. Each object has a continuous 2D position, a size, a continuous color code specified by a 3D vector in RGB space, a type specified by a one-hot vector, and a boolean unit specifying whether it is grasped. The size of the object feature vector (o) is $|o| = 39$. The agent's body has its 2D position in the environment and its gripper state (grasping or non-grasping) as features (body feature vector (b) of size $|b| = 3$). In this environment, the agent can perform various actions over the length (T) of an episode. Some of the objects (the animals) can move independently. Objects can also interact: if the agent brings food or water to an animal, it will grow in size; similarly, if water is brought to a plant, it will grow. At the end of an episode, a generative grammar generates sentences describing all the interactions that occurred. A complete specification of the environment as well as the BNF of the grammar can be found in Appendix C.1.2.

Synthetic language. To enable a controlled and systematic study of how different types of spatio-temporal linguistic meanings can be learned, we argue it is necessary to first conduct a systematic study with a controlled synthetic grammar. We thus consider a synthetic language with a vocabulary of size 53 and sentences with a maximum length of 8. This synthetic language facilitates the generation of descriptions matching behavioral traces of the agent. Moreover, it allows us to express four categories of concepts associated with specific words. Thus, the generated sentences consist in four conceptual types based on the words they involve:

- **Sentences involving basic concepts.** This category of sentences talk about present-time events by referring to objects and their attributes. Sentences begin with the *'grasp'* token combined with any object. Objects can be named after their category (eg. *'animal', 'thing'*) or directly by their type (*'dog', 'door', 'algae', etc.*). Finally, the color *('red','blue','green'*) of objects can also be specified.
- **Sentences involving spatial concepts.** This category of sentences additionally involve one-to-one spatial relations and one-to-all spatial relations to refer to objects. An object can be *'left of'* another object (reference is made in relation to a single other object), or can be the *'top most'* object (reference is made in relation with all other objects). Example sentences include *'grasp thing bottom of cat'* or *'grasp thing right most'*.
- **Sentences involving temporal concepts**. This category of sentences involves talking about temporally-extended predicates and the past tense, without any spatial relations. The two temporal predicates are denoted with the words *'grow'* and *'shake'*. The truth value of these predicates can only be decided by looking at the temporal evolution of the object's size and position respectively. A predicate is transposed at the past tense if the action it describes was true at some point in the past and is no longer true in the present, this is indicated by adding the modifier *'was'* before the predicate. Example sentences include *'was grasp red chameleon'* (indicating that the agent grasped the red chameleon and then released it) and *'shake bush'*;
- **Sentences involving spatio-temporal concepts**. Finally, we consider the broad class of spatio-temporal sentences that combine spatial reference and temporal or past-tense predicates. These are sentences that involve both the spatial and temporal concepts defined above. Additionally,

there is a case of where the spatial and the temporal aspects are entangled: past spatial reference. This happens when an object is referred to by its previous spatial relationship with another object. Consider the case of an animal that was at first on the bottom of a table, then moved on top, and then is grasped. In this case we could refer to this animal as something that was previously on the bottom of the table. We use the same *'was'* modifier as for the past tense predicates; and thus we would describe the action as *'Grasp thing was bottom of table'*.

Architectures

In this section we describe the architectures used as well as their inputs. Let one input sample to our model be $I = (S, W)$, where $(S_{i,t})_{i,t}$ represents the objects' and body's evolution, and $(W_l)_l$ represents the linguistic observations. S has a spatial (or entity) dimension indexed by $i \in [0..N]$ and a temporal dimension indexed by $t \in [1..T]$; for any i, t, $S_{i,t}$ is a vector of observational features. Note that by convention, the trace $(S_{0,t})_t$ represents the body's features, and the traces $(S_{i,t})_{t,i>0}$ represents the other objects' features. W is a 2-dimensional tensor indexed by the sequence $l \in [1..L]$; for any l, $W_l \in \mathbb{R}^{d_W}$ is a one-hot vector defining the word in the dictionary. The output to our models is a single scalar between 0 and 1 representing the probability that the sentence encoded by W is true in the observation trace S.

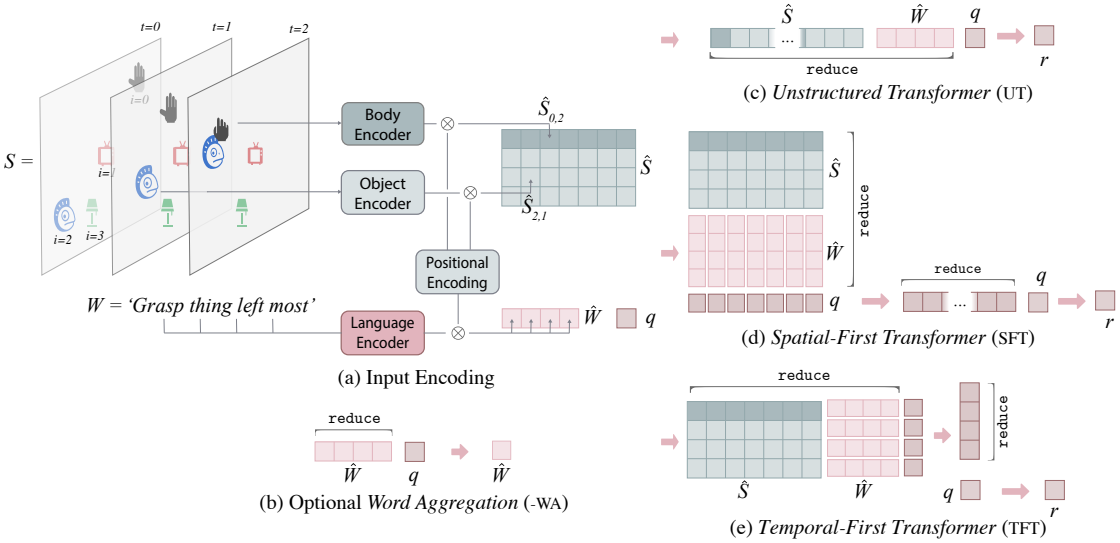


Figure 3.4: **Visual summary of the architectures used.** We show the details of UT, SFT and TFT respectively in sub (c), (d), (e), as well as a schematic illustration of the preprocessing phase (a) and the optional word-aggregation procedure (b).

Transformer Architectures. To systematically study the influence of architectural choices on language performance and generalization in our spatio-temporal grounded language context, we define a set of mapping and aggregation operations that allows us to succinctly describe different models in a unified framework. We define:

- An aggregation operation based on a Transformer model, called `reduce`. `reduce` is a parametrized function that takes 3 inputs: a tensor, a dimension tuple D over which to reduce and a query tensor (that has to have the size of the reduced tensor). R layers of a Transformer

are applied to the input-query concatenation and are then queried at the position corresponding to the query tokens. This produces an output reduced over the dimensions D.

- A casting operation called `cast`. `cast` takes as input 2 tensors A and B and a dimension d. A is flattened, expanded so as to fit the tensor B in all dimensions except d, and concatenated along the d dimension.
- A helper expand operation called `expand` that takes as arguments a tensor and an integer n and repeats the tensor n times.

Using those operations, we define three architectures: one with no particular bias (*Unstructured Transformer*, inspired by Ding et al. (2020), or UT); one with a spatial-first structural bias – objects and words are aggregated along the spatial dimension first (*Spatial-First Transformer* or SFT); and one with a temporal-first structural bias – objects and words are aggregated along the temporal dimension first (*Temporal-First Transformer*, or TFT). We provide additional explanations of the models in Appendix C.2.2.

Before inputting the observations of bodies and objects S and the language W into any of the Transformer architectures, they are projected to a common dimension. A positional encoding (Vaswani et al., 2017) is then added along the time dimension for observations and along the sequence dimension for language; and finally a one-hot vector indicating whether the vector is observational or linguistic is appended at the end. This produces the modified observation-language tuple $(\hat{S}, \hat{W})$. We let:

$$\text{UT}(\hat{S}, \hat{W}) := \texttt{reduce}(\texttt{cast}(\hat{S}, \hat{W}, 0), 0, q)$$

$$\text{SFT}(\hat{S}, \hat{W}, q) := \texttt{reduce}(\texttt{reduce}(\texttt{cast}(\hat{W}, \hat{S}, 0), 0, \texttt{expand}(q, T)), 0, q)$$

$$\text{TFT}(\hat{S}, \hat{W}, q) := \texttt{reduce}(\texttt{reduce}(\texttt{cast}(\hat{W}, \hat{S}, 1), 1, \texttt{expand}(q, N+1)), 0, q)$$

where T is the number of time steps, N is the number of objects and q is a learned query token. See Fig. 3.4 for an illustration of these architectures.

Note that SFT and TFT are transpose versions of each other: SFT is performing aggregation over space first and then time, and the reverse is true for TFT. Additionally, we define a variant of each of these architectures where the words are aggregated before being related with the observations. We name these variants by appendding -WA (word-aggregation) to the name of the model (see Fig. 3.4 (b)).

$$\hat{W} \leftarrow \texttt{reduce}(\hat{W}, 0, q)$$

We examine these variants to study the effect of letting word-tokens directly interact with object-token through the self-attention layers vs simply aggregating all language tokens in a single embedding and letting this vector condition the processing of observations. The latter is commonly done in the language-conditioned RL and language grounding literature (Chevalier-Boisvert et al., 2019; Bahdanau et al., 2019; Hui et al., 2020; Ruis et al., 2020), using the language embedding in FiLM layers (Perez et al., 2017) for instance. Finding a significant effect here would encourage using architectures which allow direct interactions between the word tokens and the objects they refer to.

LSTM Baselines. We also compare some LSTM-based baselines on this task; their architecture is described in more detail in Appendix C.2.3.

Data Generation, Training and Testing Procedures

We use a bot to generate the episodes we train on. The data collected consists of 56837 trajectories of $T = 30$ time steps. Among the traces some descriptions are less frequent than others but we make sure to have at least 50 traces representing each of the 2672 descriptions we consider. We record the observed episodes and sentences in a buffer, and when training a model we sample (S, W, r) tuples with one observation coupled with either a true sentence from the buffer or another false sentence generated from the grammar.

For each of the Transformer variants (6 models) and the LSTM baselines (2 models) we perform an hyper parameter search using 3 seeds in order to extract the best configuration. We extract the best condition for each model by measuring the mean F_1 on a testing set made of uniformly sampled descriptions from each of the categories define in section 3.3.2. We use the F_1 score because testing sets are imbalanced (the number of traces fulfilling each description is low). We then retrain best configurations over 10 seeds and report the mean and standard deviation (reported as solid black lines in Fig. 3.5 and Fig. 3.6) of the averaged F_1 score computed on each set of sentences. When statistical significance is reported in the text, it is systematically computed using a two-tail Welch's t-test with null hypothesis $\mu_1 = \mu_2$, at level $\alpha = 0.05$ (Colas et al., 2019c). Details about the training procedure and the hyper parameter search are provided in Appendix C.2.4.

3.3.3 Experiments and Results

Generalization abilities of models on non-systematic split by categories of meaning

In this experiment, we perform a study of generalization to new sentences from known observations. We divide our set of test sentences in four categories based on the categories of meanings listed in Section 3.3.2: Basic, Spatial, Spatio-Temporal and Temporal. We remove 15% of all possible sentences in each category from the train set and evaluate the F1 score on those sentences. The results are provided in Fig. 3.5.

First, we notice that over all categories of meanings, all UT and TFT models, with or without word-aggregation, perform extremely well compared to the LSTM baselines, with all these four models achieving near-perfect test performance on the Basic sentences, with very little variability across the 10 seeds. We then notice that all SFT variants perform poorly on all test categories, in line or worse than the baselines. This is particularly visible on the spatio-temporal category, where the SFT models perform at 0.75 ± 0.020 whereas the baselines perform at 0.80 ± 0.019 . This suggests that across tasks, it is harmful to aggregate each scene plus the language information into a single vector. This may be due to the fact that objects lose their identity in this process, since information about all the objects becomes encoded in the same vector. This may make it difficult for the network to perform computations about the truth value of predicate on a single object.

Secondly, we notice that the word-aggregation condition seems to have little effect on the performance on all three Transformer models. We only observe a significant effect for UT models on spatio-temporal concepts (p-value = 2.38e-10). This suggests that the meaning of sentences can be adequately summarised by a single vector; while maintaining separated representations for each object is important for achieving good performance it seems unnecessary to do the same for linguistic input. However we notice during our hyperparameter search that our -WA models are not very robust to hyperparameter choice, with bigger variants more sensitive to the learning rate.

Thirdly, we observe that for our best-performing models, the basic categories of meanings are the easiest, with a mean score of 1.0 ± 0.003 across all UT and TFT models, then the spatial ones at 0.96 ± 0.020, then the temporal ones at 0.96 ± 0.009, and finally the spatio-temporal ones at 0.89 ± 0.027. This effectively suggests, as we hypothesised, that sentences containing spatial relations or temporal concepts are harder to ground than those who do not.

Known sentences with novel observations. We also examine the mean performance of our models for sentences in the training set but evaluated on a set of *new observations*: we generate a new set of rollouts on the environment, and only evaluate the model on sentences seen at train time. We see the performance is slightly better in this case, especially for the LSTM baselines (0.82 ± 0.031 versus 0.79 ± 0.032), but the results are comparable in both cases, suggesting that the main difficulty for models lies in grounding spatio-temporal meanings and not in linguistic generalization for the type of generalization considered in this section. Plots for this experiment are reported in Appendix C.4.

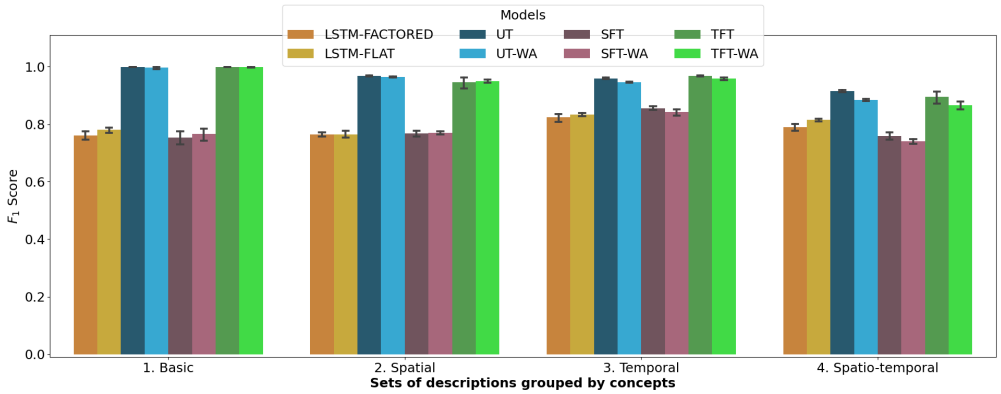


Figure 3.5: **F1 scores for all the models on randomly held-out sentences.** F_1 is measured on separated sets representing each category of concepts defined in Section 3.3.2.

Systematic generalization on withheld combinations of words

In addition to the previous generalization studies, we perform an experiment in a harder linguistic generalization setting where we systematically remove binary combinations in our train set. This is in line with previous work on systematic generalization on deep learning models (Lake & Baroni, 2018; Ruis et al., 2020; Hupkes et al., 2020). We create five test sets to examine the abilities of our models to generalize on binary combinations of words that have been systematically removed from the set of training sentences, but whose components have been seen before in other contexts. Our splits can be described by the set of forbidden combinations of words as:

1. **Forbidden object-attribute combinations.** remove from the train set all sentences containing *'red cat'*, *'blue door'* and *'green cactus'*. This tests the ability of models to recombine known objects with known attributes;
2. **Forbidden predicate-object combination.** remove all sentences containing *'grow'* and all objects from the *'plant'* category. This tests the model's ability to apply a known predicate to a known object in a new combination;

3. **Forbidden one-to-one relation.** remove all sentences containing *'right of'*. Since the *'right'* token is already seen as-is in the context of one-to-all relations (*'right most'*), and other one-to-one relations are observed during training, this tests the abilities of models to recombine known directions with in a known template;
4. **Forbidden past spatial relation.** remove all sentences containing the contiguous tokens *'was left of'*. This tests the abilities of models to transfer a known relation to the past modality, knowing other spatial relations in the past;
5. **Forbidden past predicate.** remove all sentences containing the contiguous tokens *'was grasp'*. This tests the ability of the model to transfer a known predicate to the past modality, knowing that it has already been trained on other past-tense predicates.

To avoid retraining all models for each split, we create one single train set with all forbidden sentences removed and we test separately on all splits. We use the same hyperparameters for all models than in the previous experiments. The results are reported in Fig. 3.6.

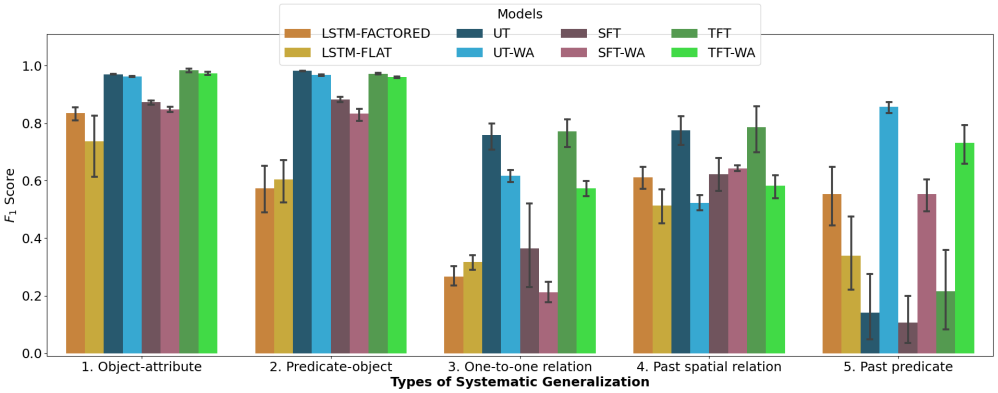


Figure 3.6: **$\mathbf{F_1}$ scores of all the models on systematic generalization splits.** F_1 is measured on separated sets representing each of the forbidden combinations of word defined above.

First we can notice that the good test scores obtained by the UT and TFT models on the previous sections are confirmed in on this experiment: they are the best performing models overall. We then notice that the first two splits, corresponding to new attribute-object and predicate-object combinations, are solved by the UT and TFT models, while the SFT models and the LSTM baselines struggle to achieve high scores. For the next 3 splits, which imply new spatial and temporal combinations, the scores overall drop significantly; we also observe much wider variability between seeds for each model, perhaps suggesting the various strategies adopted by the models to fit the train set have very different implications in terms of systematic generalization on spatial and temporal concepts. This very high variability between seeds on systematic generalization scores are reminiscent of the results obtained on the gSCAN benchmark (Ruis et al., 2020).

Additionally, for split 3, which implies combining known tokens to form a new spatial relation, we observe a significant drop in generalization for the word-aggregation (WA) conditions, consistent across models (on average across seeds, -0.14 ± 0.093, -0.15 ± 0.234 and -0.20 ± 0.061 for UT, SFT and TFT resp. with p-values $<$ 1e-04 for UT and SFT). This may be due to the fact that recombining any one-to-one relation with the known token *right* seen in the context of one-to-all relations requires a separate representation for each of the linguistic tokens. The same significant

drop in performance for the WA condition can be observed for UT and TFT in split 4, which implies transferring a known spatial relation to the past.

However, very surprisingly, for split 5 – which implies transposing the known predicate *grasp* to the past tense – we observe a very strong effect in the opposite direction: the WA condition seems to help generalizing to this unknown past predicate (from close-to-zero scores for all transformer models, the *WA* adds on average 0.71 ± 0.186, 0.45 ± 0.178 and 0.52 ± 0.183 points for UT, ST and TT resp. and p-values< 1e-05). This may be due to the fact that models without WA learn a direct and systematic relationship between the *grasp* token and grasped objects, as indicated in their features; this relation is not modulated by the addition of the *was* modifier as a prefix to the sentence. Models do not exhibit the same behavior on split 4, which has similar structure (transfer the relation *left of* to the past). This may be due to the lack of variability in instantaneous predicates (only the *grasp* predicate); whereas there are several spatial relations (4 one-to-one, 4 one-to-all).

Control experiment. We evaluate previously trained models on a test set containing hard negative examples. The aim of this experiment is to ensure that models truly identify the compositional structure of our spatio-temporal concepts and do not simply perform unit concept recognition. We select negative pairs (trajectory, description) so that the trajectories contain either the object or the action described in the positive example. Results are provided in Fig. 3.7. We observe a slight decrease of performances on all 5 categories (drop is less than 5%), demonstrating that the models do in fact represent the meaning of the sentence and not simply the presence or absence of a particular object or predicate.

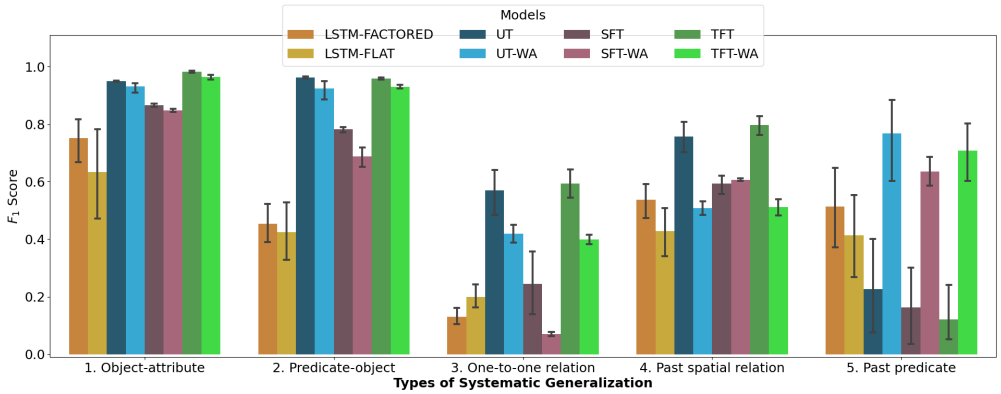


Figure 3.7: **Control experiment: F1** scores for all the models on systematic generalization splits in the hard negative examples setting.

3.3.4 Related Work

The idea that agents should learn to represent and ground language in their experience of the world has a long history in developmental robotics (Zwaan & Madden, 2005; Steels, 2006; Sugita & Tani, 2005; Cangelosi et al., 2010) and was recently extended in the context of Language Conditioned Deep Reinforcement Learning (Chevalier-Boisvert et al., 2019; Hermann et al., 2017; Luketina et al., 2019; Bahdanau et al., 2019). These recent approaches often consider navigation (Chen & Mooney, 2011; Chaplot et al., 2018) or object manipulation (Akakzia et al., 2021; Hermann et al.,

2017) tasks and are always using instructive language. Meanings typically refer to instantaneous actions and rarely consider spatial reference to objects (Paul et al., 2016). Although our environment includes object manipulations, we here tackle novel categories of meanings involving the grounding of spatio-temporal concepts such as the past modality or complex spatio-temporal reference to objects.

We evaluate our learning architectures on their ability to generalise to sets of descriptions that contain systematic differences with the training data so as to assess whether they correctly model grammar primitives. This procedure is similar to the *gSCAN* benchmark (Ruis et al., 2020). This kind of compositional generalisation is referred as 'systematicity' by Hupkes et al. (2020). Environmental drivers that facilitate systematic generalization are also studied by Hill et al. (2020). Although Hupkes et al. (2020) consider relational models in their work, they do not evaluate their performance on a *Language Grounding* task. Ruis et al. (2020) consider an Embodied Language Grounding setup involving one form of time-extended meanings (adverbs), but do not consider the past modality and spatio-temporal reference to objects, and do not consider learning truth functions. Also, they do not consider learning architectures that process sequences of sensorimotor observations. To our knowledge, no previous work has conducted systematic generalization studies on an Embodied Language Grounding task involving spatio-temporal language with Transformers.

The idea that relational architectures are relevant models for Language Grounding has been previously explored in the context of *Visual Reasoning*. They were indeed successfully applied for spatial reasoning in the visual question answering task *CLEVR* (Santoro et al., 2017b). With the recent publication of the video reasoning dataset *CLEVRER* (Yi et al., 2019), those models were extended and demonstrated abilities to reason over spatio-temporal concepts, correctly answering causal, predictive and counterfactual questions (Ding et al., 2020). In contrast to our study, these works around CLEVRER do not aim to analyze spatio-temporal language and therefore do not consider time-extended predicates or spatio-temporal reference to objects in their language, and do not study properties of systematic generalization over sets of new sentences.

3.3.5 Discussion and Conclusion

In this work, we have presented a first step towards learning Embodied Language Grounding of spatio-temporal concepts, framed as the problem of learning a truth function that can predict if a given sentence is true of temporally-extended observations of an agent interacting with a collection of objects. We have studied the impact of architectural choices on successful grounding of our artificial spatio-temporal language. We have modelled different possible choices for aggregation of observations and language as hierarchical Transformer architectures. We have demonstrated that in our setting, it is beneficial to process temporally-extended observations and language tokens side-by-side, as evidenced by the good score of our Unstructured Transformer variant. However, there seems to be only minimal effect on performance in aggregating temporal observations along the temporal dimension first – compared to processing all traces and the language in an unstructured manner – as long as object identity is preserved. This can inform architectural design in cases where longer episode lengths make it impossible to store all individual timesteps for each object; our experiments provide evidence that a temporal summary can be used in these cases. Our experiments with systematic dimensions of generalization provide mixed evidence for the influence of summarizing individual words into a single vector, showing it can be detrimental to generalize to novel word combinations but also can help prevent overgeneralization of a relation between a single word and a single object without considering the surrounding linguistic context.

Limitations and further work. There are several limitations of our setup which open important opportunities for further work. First, we have used a synthetic language that could be extended: for instance with more spatial relations and relations that are more than binary. Another axis for further research is using low-level observations. In our setting, we wanted to disentangle the effect of structural biases on learning spatio-temporal language from the problem of extracting objects from low level observations (Matthey et al., 2019; Greff et al., 2020; Engelcke et al., 2020; Locatello et al., 2020; Carion et al., 2020) in a consistent manner over time (object permanence (Creswell et al., 2020; Zhou et al., 2021)). Further steps in this direction are needed, and it could allow us to define richer attributes (related to material or texture) and richer temporal predicates (such as breaking, floating, etc). Finally, we use a synthetic language which is far from the richness of the natural language used by humans, but previous work has shown that natural language can be projected onto the subspace defined by synthetic language using the semantic embeddings learned by large language models (Marzoev et al., 2020): this opens up be a fruitful avenue for further investigation.

3.4 Chapter Discussion

Overall, in this chapter we (in Section 3.2.2) have motivated the use of the reward learning problem as a first lens to study the possibility of multimodal grounding for autotelic agents acting in an embodied, non-linguistic world by following language goals. In the first study, we have cast the goal-conditioned reward learning problem as the problem of learning classes of equivalence of spatial configurations of objects. This task was selected for two reasons: first, it explicitly has the relational structure that we wanted to test in this chapter; and second the dataset this model is trained on seems pretty natural: an embodied agent observing configurations of objects as its perspective changes and the objects stay fixed. In this setting thus, the goal-matching task represented by the Discrimination subtask is especially relevant to study goal-conditioned reward functions, and the Identification task provides us with additional information about the structure of the problem. We conclude from our study that relational architectures, be it Deep Sets, Recurrent Deep Sets, GCNs or Message-Passing GNNs, have an advantage compared to the less structured but much more parameter-rich ResNet18. Within the space of relational architectures, only the MPGNN reaches a relatively high accuracy, and the differences between GNN type only accentuate with the number of objects considered.

After establishing these results, in Section 3.3 we moved towards a true multimodal setting featuring a simplified language which includes spatial relations, temporal relations and the past tense in addition to more traditional predicates and objects. Some of these predicates, like *grow* are temporal in nature since they involve following the size of an object accross multiple time steps. In this *Temporal Playground*, we aim to learn a reward function predicting the co-occurence of sentences of this language and the words they refer to. To this aim we introduced several architectural variants and established that although the Unstructured Transformer variant performs best overall, performance is still mostly retained in the Temporal-first transformer, suggesting that compression along the time dimension is a reasonable architectural choice in this setting. This highlights the importance of object permanence (and of relational inductive biases, since a Transformer on temporally-extended objects is very close in structure to an MPGNN) in spatio-temporal grounding and should inform our efforts in building multimodal autotelic agents.

Nevertheless, we should recall that what we have studied in this section is only one of the aspects of multimodal grounding. The systematic generalization properties of the Transformer reward seem good enough so that they can be used to train a language autotelic agent in an embodied environment

on language goals that it has never seen before. Within this setup, the agent could collect additional trajectories which could be hindsight relabelled by the Social Peer, as in the ScienceWorld work of Section 2.2.

However, moving towards more realistic image settings and natural language requires us to use pretrained vision language models. Indeed, there is little hope to achive perfect grounding for arbitrary images and language from synthetic supervision or small datasets alone: prior knowledge will need to incorporated in such reward functions. Current prominent pretrained vision-language models (for instance Alayrac et al. (2022)) have a structure resembling the unstructured transformer of Contribution 4, with the important difference that today's widespread models incorporate no architectural priors about objects or their relations. Unfortunately these vision-language models are not always open-source (thus easily usable). Even when such a model is made public its systematic generalization capabilities is not always clear. Take for instance the MineCLIP reward of the MineDojo work (Fan et al., 2022). This is a time-extended version of CLIP (Radford et al., 2021) that has been pretrained on a language-video dataset of Minecraft lets plays collected from Youtube. Though the authors demonstrate that this reward can be used to train relatively simple goal-conditioned policies, with the goals expressed in language, this does not scale to the more complex language goals that could be expressed in Minecraft. Pretrained vision-language models themselves do not have that good of a generalization ability (Du et al., 2023a). Maybe incorporating relational biases in the multimodal model architecture could help, by encouraging the model to preserve information in recombinable chunks. This could be done in the natural way, through architecture, or in a more contrived way, by using pretrained object extraction networks and training on the resulting object vectors.

Beyond grounding through reward learning alone, recall that an autotelic language-conditioned embodied agent needs to be able to sample its own goals so that they are reasonable in its current environment (conditioned on the first observation for instance) and interesting considering the agent's previous skills. For sample efficiency reasons we must also ideally be able to provide hindsight feedback to the agent, which is the description problem mentioned in the chapter background. Captioning images in general is still a hard problem, and for complex images the models are prone to hallucination. As for the goal-conditioned policy learning part in itself, this is also a hard problem in general, especially in the highly multitask settings required of an autotelic agent. Training RL agents with an important number of goals from scratch is very difficult (the ScienceWorld autotelic agent of Section 2.2 has trouble generalizing to other goals, and the reason why the LMA3 agent doesn't use RL is that it was too sample-inefficient for the amount of goals we needed it to learn). Massively multitask language-conditioned policies are still a problem, especially in a multimodal setting. Additionally, for the grounding part, usually low-level control with RL is still difficult. All of this makes learning truly general goal-conditioned policies very hard, especially in the multimodal setting.

Considering all the previous arguments, we are still far from being able to implement truly open-ended language-conditioned agents. True open-ended behavior is unreachable in simple environments (as the limitations on agent behavior in Chapter 2 have shown), and complex environments rely on pretrained models who are expensive and not yet perfect. To be more specific, ideally to build a true embodied autotelic agent exploring the space of skills that can be described with language, we need:

- A context-aware goal generation module (able to condition on first state and previous agent skills) – this seems within reach of state-of-the-art generative vision-language models such as GPT4, albeit while not being perfect;

- A systematically generalizing language conditioned, temporally extended reward function: this is still problematic (and the time horizon needed is very large for general skills). Might an architecture based on object persistence as in Contribution 4 help?
- A general-purpose, continually-learning massively multitask goal-conditioned policy. This seems to be very difficult to train (but see the work of Team et al. (2023) for ideas involving an open-ended goal repertoire and continual learning through policy distillation);
- A goal relabeler that is able to take a trajectory as input and output sentences describing the trajectory that are both true and nontrivial such as the agent can learn skills from them. This seems harder than the goal generation (less emphasis on creativity and more on the correctness of the goal function), but is not strictly necessary if the goal generation and reward are good enough and the generated goals have a high chance of being accomplished by an imperfect agent.

Given these points and the scale at which these models must be trained and run, the endeavour seems quite difficult, and the environment itself must be pretty large for real open-ended behaviors to be learned, which complicates manners further. However, recall our argument from the introduction of Chapter 2. We argued that language itself defined an environment of arbitrary complexity. In light of the previous arguments, it seems that carrying on research on intrinsically-motivated open-ended agents in textual environments only might yield more insights on the processes of intrinsic motivation than agents operating in embodied environments. All that remains is to find an open-ended text domains allowing for verifiable, open-ended behaviors and ideally on which language models might have some priors we can re-use.

Chapter 4

Code-Generating Autotelic Agents

Preamble to the chapter

In this final experimental chapter, we develop an insight that solves both the issues of working with open-ended environments and of grounding. The possibilities of programming are endless, and goals can be expressed for autotelic agents as unit tests. Program execution provides a ground truth against which the agent can train, allowing us to sidestep the problem of grounding by learning reward functions (as in Chapter 3). Building on this insight we propose two contributions. The first (Section 4.2) introduces ACES, a method for generating a diversity of programming puzzles, based on a categorization of programming puzzles as involving a particular combination of programming skills. Both generation of new puzzles and the prediction of their category is done with an LLM. We show that the use of this combinatorial semantic space leads to the generation of a larger diversity of puzzles, on a variety of measures. We move on to our second contribution of the chapter (Section 4.3), where we cast autotelic learning in the domain of programming puzzles as a game between two agents: a Setter generating hard, but still solvable, puzzles, and a Solver learning to solve them. We call this game Codeplay. In first experiments, we show that the Setter, a smaller language model finetuned with reinforcement learning, manages to generate puzzles that are hard, solvable, and relatively novel. We conclude by discussing the numerous avenues for further work these contributions open. The Sections 4.2 and 4.3 reuse material from the following contributions:

- **ACES: generating diverse programming puzzles with autotelic language models and semantic descriptors**; Julien Pourcel*, Cédric Colas*, Pierre-Yves Oudeyer, Laetitia Teodorescu; *Under review for the International Conference on Learning Representations, 2024*; This collaboration came from the internship of Julien, on the original project to investigate goal-creation processes from examples. Laetitia initiated the project, Cédric, Laetitia an Julien devised experiments, Julien ran experiments, Julien, Cédric and Laetitia ran analyses, Laetitia, Pierre-Yves and Cédric advised, Julien, Cédric and Laetitia wrote the paper.
- **Codeplay: Autotelic Learning through Collaborative Self-Play in Programming Environments**; Laetitia Teodorescu, Cédric Colas; Thomas Carta, Matthew Bowers, Pierre-Yves Oudeyer, *In preparation, prensented at the Intrinsically-Motivated Open-Ended Learning Conference, 2023*; This collaboration, initiated by Laetitia, additionally acknowledges discussions with Patrick Haluptzok and Adam Tauman Kalai.

4.1 Introduction to the chapter

For this chapter, we will not have a detailed background discussion, but rather shortly explain how to try to overcome the limitations of our approaches in the previous chapters. If you recall our discussion in Chapter 3, we have concluded that techniques used in grounding language to a sensorimotor environment might be limited by today's techniques. While there has been a recent surge in work around multimodal models, these models are still imperfect and most of them are not as easily available as LLMs; they may additionally not always transfer to the domain we want to use them on. However there is a way to precisely interact with an embodied environment: use calls to APIs as in the work of Wang et al. (2023a). Programming might be the solution in this case: programs are open-ended and compositional: they define a perfect space for autotelic exploration; programs and their execution define concrete tasks with a clearly-defined notion of success and failure, which means that an agent can invent creative, freeform goals in the space of programs without being constrained by small, text-based environments as was the case in our contributions in ScienceWorld and LMA3 in the smaller CookingWorld in our Contribution 2.3.

4.1.1 Python Programming puzzles and the P3 dataset.

The space of all programs is too vast for our early investigations. Working on an instruction-code setup, where an instruction is provided in natural language and must be implemented as code, such as in the HumanEval benchmark for code generation (Chen et al., 2021b), would map very well to a goal-conditioned program synthesis agent. However this would require the use of an LM-based task-completion function to determine whether an instruction was correctly followed (as in LMA3). We thus turn to another domain, that is at once expressive and that has a more rigorous structure: programming puzzles.

The *Python Programming Puzzles dataset* (P3) contains 1715 puzzle-solution pairs where each puzzle is defined by a test program `f` designed to verify the validity of solution programs `g` such that valid solutions satisfy `f(g()) == True` when run in an interpreter, see an example for the classical Towers of Hanoi in Figure 4.7 (Schuster et al., 2021). P3 puzzles span problems of various difficulties that involve different programming skills: e.g. string manipulation, classical, more complex programming problems (e.g. involving dynamic programming or factoring), puzzles from the mathematics Olympiad, or even open problems in computer science or mathematics. The P3 dataset is split into training and testing datasets ($N = 636$ and 1079 respectively). Programming puzzles thus represent an expressive subspace of python programs, and the correctness of a solution for a given puzzle can automatically be checked.

Contents of this chapter In the two contributions that follow, we use the P3 domain to implement code-producing autotelic systems. Our first system, ACES, is conceptually close to quality-diversity methods or population-based autotelic algorithms (Mouret & Clune, 2015b; Pugh et al., 2016). The second contribution, still in progress, introduces Codeplay which is an algorithm that casts autotelic learning as a collaborative game between two agents, in the manner of asymmetric self-play (Sukhbaatar et al., 2018).

```python
def f(moves: List[List[int]]):
    """
    Eight disks of sizes 1-8 are stacked on three towers, with
       each tower having disks in order of largest to
    smallest. Move [i, j] corresponds to taking the smallest
       disk off tower i and putting it on tower j, and it
    is legal as long as the towers remain in sorted order. Find
       a sequence of moves that moves all the disks
    from the first to last towers.
    """
    rods = ([8, 7, 6, 5, 4, 3, 2, 1], [], [])
    for [i, j] in moves:
        rods[j].append(rods[i].pop())
        assert rods[j][-1] == min(rods[j]), "larger␣disk␣on␣top␣
           of␣smaller␣disk"
    return rods[0] == rods[1] == []

def sol():
    # taken from https://www.geeksforgeeks.org/c-program-for-
       tower-of-hanoi/
    moves = []
    def hanoi(n, source, temp, dest):
        if n > 0:
            hanoi(n - 1, source, dest, temp)
            moves.append([source, dest])
            hanoi(n - 1, temp, source, dest)
    hanoi(8, 0, 1, 2)
    return moves

assert f(g()) is True
```

Figure 4.1: Example of a simple programming puzzle and its solution from the P3 dataset (Schuster et al., 2021). A solution function `g` must return a valid solution such that `f(g()) is True`.

4.2 Contribution 5: ACES – Generating diverse programming puzzles with autotelic language models and semantic descriptors

4.2.1 Introduction

Finding and selecting new and interesting problems to solve is at the heart of curiosity, science and innovation (Chu & Schulz, 2020; Schmidhuber, 2013; Herrmann et al., 2022). We propose to leverage machine learning, a set of tools usually targeted at *solving problems*, to automate the generation of an *interesting diversity of solvable problems*. Automated problem generation has a wide range of applications such as education (generating problems for students to solve), data augmentation (generating problems and solutions for AI model training), or automated scientific discoveries (e.g. discovering new scientific problems and their solutions).

In this work, we focus on the generation of a diversity of Python programming puzzles, an open-ended space to explore that contains problems ranging from trivial string manipulations to

open mathematical puzzles (Schuster et al., 2021). Importantly, puzzle-solution pairs produced by the search can be checked for correctness using a Python interpreter, providing a notion of ground truth that natural language problems lack (e.g. creative writing). The automated generation of diverse programming puzzles could benefit computer science education and be used as a data generation process for the training of large language models (LLMs). Pretraining on code indeed seems to be a major factor in LLMs' reasoning abilities (Madaan et al., 2022; Liang et al., 2022b; Fu et al., 2022).

Standard generative models do not explicitly optimize for diversity but are instead trained to fit the distribution of a reference dataset (e.g. Goodfellow et al., 2014; Brown et al., 2020; Chen et al., 2020a; Ho et al., 2020). Measuring and optimizing for diversity requires the definition of a *behavioral characterization* (BC) of the generated artefacts on which to evaluate the measure. Early diversity-producing methods often used hand-coded low-dimensional representation functions, which focused and restricted the diversity search along features one could easily compute. More recent methods leverage pretrained embedding functions allowing them to work with higher-dimensional data (e.g. image, text, programs) at the expense of interpretability and control over the axes of variations (Nair et al., 2018; Laversanne-Finot et al., 2018; Cully, 2019; Etcheverry et al., 2020).

We propose to leverage *semantic descriptors*: a hand-defined list of abstract features evaluated by LLMs. Semantic descriptors allow to work with high-dimensional inputs (here programs) while focusing the diversity search along interpretable semantic features of interest. Specifically, we represent any puzzle by the set of programming skills required to solve it among 10 possible skills (e.g. graphs, dynamic programming, recursion). Evaluating descriptors with LLMs allows us to define more abstract features that better capture our intuitive perception of axes of variation, descriptors that would have been hard or even impossible to code by hand. The compositionality of language and linguistic categories futher allow us to easily define sets of orthogonal conceptual categories that can be almost arbitrarily combined (Colas et al., 2020b).

This work introduces a new diversity-producing algorithm called ACES for *Autotelic Code Exploration with Semantic descriptors*. ACES leverages an LLM for puzzle generation, solution generation and novelty evaluation. It slowly grows an *archive* of discovered puzzle-solution pairs, where each cell of the archive contains puzzles that share a given semantic representation — a 10D binary vector obtained by the semantic descriptors. At each new cycle of the algorithm, ACES targets a cell randomly in the archive (semantic goal) and generates a candidate puzzle and solution by prompting the LLM-based puzzle generator with the target semantic representation and a set of examples from the archive. The generated puzzle-solution pair is then evaluated for validity using a Python interpreter and, if valid, gets encoded by the puzzle labeler into a corresponding semantic representation used to store the newly discovered puzzle in the right archive cell, see Figure 4.2. Our experiments study the evolution of several diversity metrics over time and compares ACES with state-of-the-art baselines (Lehman et al., 2022; Haluptzok et al., 2023).

To summarize, our contributions in this work are the following:

- We define the notion of *semantic descriptors* to leverage LLMs for the encoding of high-dimensional textual data into hard-to-compute, abstract and interpretable features of interest.
- We introduce a set of such semantic descriptors to characterize the diversity of programming puzzles based on classical programming ontologies.
- We propose *Autotelic Code Exploration with Semantic descriptors* (ACES), a new diversity-producing method building on these semantic descriptors that leverages the few-shot learning abilities of LLMs to generate an interesting diversity of programming puzzles;

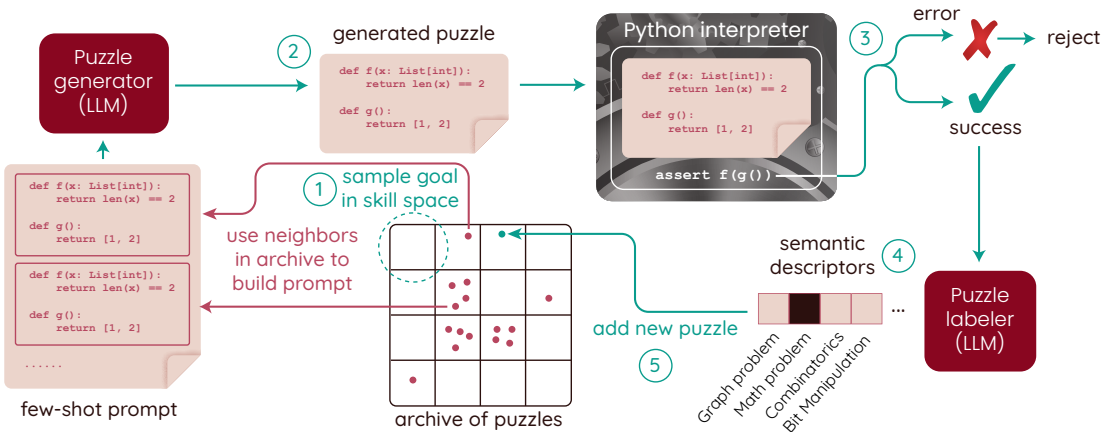


Figure 4.2: Overview of ACES. ACES maintains an archive of dcreate figure from tcolorboxiscovered puzzles grouped into cells indexed by their semantic representation (skill combination). ACES runs in several steps: 1) sample a target semantic goal and relevant examples from the archive. 2) given these, generate a puzzle `f` and its solution `g` with the puzzle generator. 3) test the validity of that pair by running `assert(f(g()))` in the interpreter. 4) if the pair is valid, obtain its semantic representation with the puzzle labeler. 5) add the new pair to its corresponding cell in the archive.

- We evaluate the ability of ACES and its baselines to achieve various kinds of diversities and provide a comprehensive analysis of the interactions between diversity and finetuned performance on a held out test set.

4.2.2 Related Work

Diversity-producing algorithms were originally proposed within the field of evolutionary computing. Beginning with *novelty search* (Lehman & Stanley, 2011c,b), this line of research expanded with the invention of quality-diversity algorithms (QD: Mouret & Clune, 2015a; Cully & Demiris, 2018a), a set of methods striving to evolve a diverse population of locally-performant individuals via the undirected mutation of existing solutions. A parallel line of research introduced *goal-directed* exploration processes, also called *autotelic learning*, where exploring agents learn to represent and sample their own goal as a way to direct the diversity search (Baranes & Oudeyer, 2013c; Forestier et al., 2022b; Colas et al., 2022c). Although autotelic methods were first developed to model the open-ended development of children in skill learning robots (**??**), they have also proved effective in the automatic exploration of complex systems, either simulated (Reinke et al., 2019; Etcheverry et al., 2020) or physical (Grizou et al., 2020).

In all these methods, one must define a BC space to characterize novelty. The earliest works used predefined low-dimensional descriptors to represent generated artefacts (Lehman & Stanley, 2011c; Baranes & Oudeyer, 2013c; Mouret & Clune, 2015b), which constrains the search along a handful of features one can code a descriptor for. More recent works have relied on higher-dimensional learned or pretrained embedding functions (Nair et al., 2018; Laversanne-Finot et al., 2018; Reinke et al., 2020), and even hierarchies of such spaces, each representing different perceptual features of the generated artefacts (Cully & Demiris, 2018b; Etcheverry et al., 2020). Diversity-search algorithms sometimes need to be adapted to work with such high-dimensional spaces whose discretization leads to an exponential number of cells (Vassiliades et al., 2017). But the main issue is that they are hardly

interpretable and might not always align with the dimensions of variation humans find meaningful. With ACES, we propose an autotelic diversity-producing algorithm that constrains the search along a set of abstract, interpretable and hard-to-compute features of interest evaluated by LLMs.

This work follows of a recent trend on leveraging feedback or generations from LLMs to improve older learning architectures. The original inspirations for this contribution come from the QD literature, where LLMs are now used to suggest mutations and crossover in diversity-producing evolutionary algorithms (Lehman et al., 2022; Bradley et al., 2023b; Meyerson et al., 2023). Several approaches also rely on LLMs to replace human feedback (AI feedback): e.g. to finetune other LLM models (Bai et al., 2022; **?**), to characterize generated poems (Bradley et al., 2023a), to revise the policy of autotelic agents (Wang et al., 2023a), to suggest goals for them (Colas et al., 2023; Du et al., 2023b) or measure their *interestingness* (Zhang et al., 2023). With ACES, we use AI feedback to compute abstract and intepretable representations of programming puzzles so as to optimize for diversity in that space. In a similar way. QDAIF (parallel work) uses 2D LLM-generated characterisations in the context of poem generation, a space where there is not such a clear notion of feasibility and solvability (Bradley et al., 2023a).

4.2.3 Methods

We first discuss measures of *interesting diversity* (Section 4.2.4). Then, we introduce ACES, our new method for diversity generation (Section 4.2.5) and define relevant baselines (Section 4.2.6). We will open-source the implementation of the algorithms and the datasets of generated puzzles and solutions with the camera-ready version.

4.2.4 Measuring interestingness and diversity

Any generative process aims to generate collections of artefacts that are both *diverse* and *interesting*. Because these are *subjective* measures that strongly depend on the observer's point of view, we must define them carefully.

Defining interesting representation spaces. One way to measure interesting diversity is to compute the diversity of a collection of artefacts in a representation space where *everything is interesting*, meaning that all uninteresting samples collapse to the same point. This requires the careful definition of a *representation function* R mapping each artefact p (here puzzle) to a numerical representation $z_p = R(p)$ and a *metric* m to compute distances between them. What should these functions be in the context of our programming puzzles?

First, we use cosine distance computed in three different continuous embedding spaces — a standard approach for representing programs and text in general (Reimers & Gurevych, 2019). Here, we use *salesforce/codet5p-110m-embedding* (Wang et al., 2023b), *wizardlm/wizardcoder-1b-v1.0* and *wizardlm/wizardcoder-3b-v1.0* (Luo et al., 2023) embedding models from the HuggingFace's Hub (Wolf et al., 2020) to obtain 256D, 2048D, and 2816D continuous embedding representation vectors.

Second, we propose to represent programming puzzles using *semantic descriptors* — a set of hand-defined labels that may align better with the way humans perceive the ontology of programming puzzles. We define 10 semantic labels: Sorting and Searching, Counting and Combinatorics, Tree and Graph problems, Mathematical Foundations, Bit Manipulation, String Manipulation, Geometry and

Grid Problems, Recursion and Dynamic Programming, Stacks and Queues, Optimization Algorithms, see definitions in Appendix Section D.1.2. A puzzle is then represented as a 10D binary semantic vector z_p, where each value z_p^i evaluates whether the puzzle requires skill s_i (1) or not (0) to be solved. Writing code to encode puzzles in this way seems highly non-trivial. Instead, we ask ChatGPT to compute them (version *gpt-3.5-turbo-0613*). In the example of 4.7 for instance, ChatGPT detects the need for *Sorting and Searching*, *Counting and Combinatoris* as well as *String Manipulation* and thus encodes the puzzle as 1100010000 (see other examples in Appendix Section D.1.2). We use Hamming distance in that semantic space.

This semantic representation function is a contribution of this contribution. We hypothesize that it is more aligned with human perception of programming puzzles than continuous embedding functions. Our proposed diversity generation process is designed to maximize this form of diversity (see Section 4.2.5). We hypothesize that this will achieve not only higher interesting diversity scores in this semantic representation space, but also higher scores in the continuous embedding representation spaces.

Measuring diversity. We use several metrics to measure the diversity of a set of discovered puzzles. We first use counts of discovered puzzles and cells: 1) the number of cells that were discovered (at least 1 puzzle was found in the cell), 2) the number of cells discovered beyond the ones covered by the train set, 3) the number of valid puzzles that were generated, 4) the number of valid puzzles generated beyond the cells covered by the train set. Second, we track measures of density or entropy: 5) the average pairwise distance between embedding representations, 6) the entropy of the distribution of semantic representations.

We use the *Vendi score*, a measure that extends ideas from ecology and quantum statistical mechanics to machine learning (Friedman & Dieng, 2022). The Vendi score is a domain-agnostic diversity metric that is mathematically defined as the exponential of the Shannon entropy of the eigenvalues of a similarity matrix K, whose entries are equal to the similarity scores between each pair of elements in the collection. It does not assume access to a reference distribution and it is symmetric to the order of elements in the collection and to the order of features in their representations. It can be thought of as computing an *effective number* of dissimilar elements in the collection, an approach that has been extensively argued for in ecology (Hill, 1973; Patil & Taillie, 1982; Jost, 2006), economics (Adelman, 1969) and machine learning (Friedman & Dieng, 2022). The Vendi score is applied on the two pairs of representation and similarity functions described above.

An utilitarian take on measuring interesting diversity. Interesting puzzles must be solvable, which is why we filter out invalid puzzle-solution pairs. One could also be interested in training a puzzle solver to achieve high-performance on a specific problem distribution. In this case, we would perceive a collection of generated puzzle-solution pairs as *more interesting than others* if the solver finetuned on this set outperforms the same solver finetuned on the other sets when tested on the target distribution (e.g. P3's test set). Section 4.2.7 will look at correlations between various metrics and the final performance of a LLaMA model (*openlm-research/open_llama_3b_v2* on HF's hub, Geng & Liu, 2023) after finetuning for two epochs on the generated set of puzzle-solutions. In line with previous research (Chen et al., 2021b), we will report the Pass@k performance metric on the testing set of P3 for $k \in [1:10]$: the percentages of puzzles for which at least one valid solution is generated within k attempts. In addition to the diversity metrics listed above, we will look at various metrics measuring how well the generated set of puzzle-solution pairs covers the testing distribution.

4.2.5 Autotelic generation of interesting diverse puzzles

This section introduces ACES, a new diversity-producing algorithm that generates an interesting diversity of programming puzzles by optimizing for the novelty of each new generation in the semantic space described above, see Figure 4.2. ACES grows an *archive* of diverse puzzle-solution pairs. It repeatedly: samples a semantic goal from the archive, generates a new puzzle-solution pair conditioned on that goal and, if the pair is valid, labels and adds it to the archive. We use the *ChatGPT* LLM (*gpt-3.5-turbo-0613,* Schulman et al., 2022). Algorithm 1, Figure 4.2 and Appendix Section D.1 respectively present the pseudo-code, illustration and prompts of ACES.

Algorithm 1 Pseudo-code of ACES

Initialize an archive $\mathcal{A}$ (with labeled puzzle-solution pairs from the P3 train set)

for $i = 1\ N$ **do** Sample a goal: $z_g \sim \text{Uniform}(\mathcal{A})$ (uniform sample of a semantic goal)
Sample examples: $e \sim E(\mathcal{A}, z_g)$ (nearest neighbor sampling with Hamming distance)
Generate puzzle and solution: $(p,\ s) \sim \text{LLM}(\text{prompt}_{\text{gen}}(g,\ e))$ (see Appendix D.1)
Test puzzle-solution pair: pass = `p(s())==True` (using the interpreter)

if pass **then** Label the puzzle $z_p \sim \mathcal{LLM}(\text{prompt}_{\text{lab}},\ p)$ (see Appendix D.1)
Add $(p,\ s,\ z_p)$ to the archive $\mathcal{A}$ in cell c_{z_p}

Sampling a goal and relevant examples.ACES selects a semantic goal by sampling uniformly among the set of 2^{10} possible semantic representations. We then greedily select three closest examples from the archive using the Hamming distance in the semantic space: two from the generated puzzle-solution pairs and one from P3's train set to always keep an well-formatted puzzle example.

Puzzle generator. The puzzle generator is implemented by an LLM. Conditioned on the semantic goal and the three examples, we ask the LLM to generate a puzzle-solution pair that would be labeled with the target semantic vector. For each *cycle* of the algorithm, we repeat the process of sampling a goal, examples, puzzles and solutions 10 times before considering the addition of these candidates to the archive. This is done to leverage parallel calls to the LLM API and could be done with 1 generation per cycle. Using a Python interpreter, we filter the set of valid puzzle-solution pairs and send them to the puzzle labeler.

Puzzle labeler. The puzzle labeler computes the semantic representation vector of each valid puzzle-solution pair. The prompt details the task to the LLM and presents the complete list of skills, then asks it to compute its semantic representation (see Section 4.2.4). The puzzle-solution pair is finally added to its corresponding cell in the archive. Note that, although we aim for a particular target semantic representation, whether we achieve the goal or not is not that important. What is important is that the generate puzzle is valid and falls into a new cell.

What's new?In addition of the semantic descriptors (whose novelty is discussed in Section 4.2.4), ACES is the first algorithm to leverage an autotelic LLM for the generation of diverse artefacts via in-context learning. The LLM is here used as the generation engine and is steered towards generating novel and interesting puzzles via its goal-directedness and example selection (in-context learning).

The algorithm *ELM* already used an LLM to suggest mutations of existing artefacts (Lehman et al., 2022). But like other quality-diversity algorithms, it is not goal-directed: it samples a previously-generated artefact from its archive, mutates it with the LLM in the hope of generating a new artefact that would fill a new cell.

4.2.6 Baselines

Static generative model (Static Gen). This baseline was proposed in Haluptzok et al. (2023): it repeteadly prompts the LLM to generate a new puzzle-solution pair given three examples uniformly sampled from P3's train set.

Ablation of goal-directedness (ELM semantic). Instead of sampling a goal and asking the puzzle generator to reach it, we uniformly sample two puzzle-solution pairs from the archive that serve as examples. We then sample a cell in the archive and a puzzle from that cell, and ask the language model to output a mutation of this puzzle. The resulting algorithm is not autotelic anymore but becomes a variant of the QD algorithm *Map-Elites* (Mouret & Clune, 2015b). In fact, this implementation is a variant of the *ELM* algorithm (Lehman et al., 2022) where the explored representation space is our proposed semantic space.

Ablation of goal-directedness and semantic representations (ELM). We can further ablate ACES by removing the use of the semantic representation space. Instead, this baseline uses the continuous embedding space described in Section 4.2.4 (*Salesforce/codet5p-110m-embedding*, Wang et al., 2023b). This ablation is a variant of ELM (Lehman et al., 2022) where the explored representation space is a pretrained embedding space. To define a limited number of cells in this high-dimensional space, we use the method proposed in *CVT-Map-Elites*, a variant of MapElites that uses centroidal Voronoi tessallations (CVTs) to partition the space into a tractable number of well-distributed cells (Vassiliades et al., 2017). The partition is conducted in two steps. We first sample with replacement 40k puzzles from P3's train set and perturb their embeddings with a Gaussian noise ($\mathcal{N}(\mu = 0, \sigma^2 = 1.2)$) before normalizing them to unit-length. Then, we use the K-means algorithm (Steinhaus, 1957; MacQueen, 1967) to identify 1024 centroids and obtain the same number of cells as ACES in the archive. Once this archive is created, we simply run the ELM algorithm. The difference between ELM-semantic and ELM is the definition of the archive.

Generating an interesting diversity of textual artefacts. Although the current paper focuses puzzle generation, ACES can in principle be used to generate an interesting diversity of any type of textual artefacts. For each new type of textual artefacts we want to generate a diversity of, we need to provide a set of features of interest the LLM will be able to evaluate on each new generation. Natural language provides a rich and flexible descriptive space. Compared to traditional representation functions that rely on hand-engineered features, the abstract nature of language allows us to describe the semantic qualities of textual artefacts in an open-ended way.

4.2.7 Results

Is llm labeling faithful?

Our proposition of leveraging LLMs to semantically characterize generated programming puzzles is only meaningful to the extent that the LLM faithfully labels the puzzles. To make sure this is the case on the distribution of puzzles we end up generating, we compare the LLM-generated labels to hand-defined labels on a set of 60 puzzles sampled from the generated set of the seed of ACES which has the highest label diversity. Details of how the puzzles were sampled for the computation of the confusion matrix can be found in Appendix Section D.2.

Figure 4.3 reports the confusion matrix and the number of puzzles containing the ground truth label for each row. The puzzle labeler demonstrates high true positive rates on most dimensions but sometimes struggles to detect present skills (true nagative rate), e.g. for the Stacks and Queues and the Geometry and Grid dimensions. Note that annotating puzzle with semantic labels is also hard for humans and that the classification does not need to be perfect to drive diversity (see Section 4.2.7).

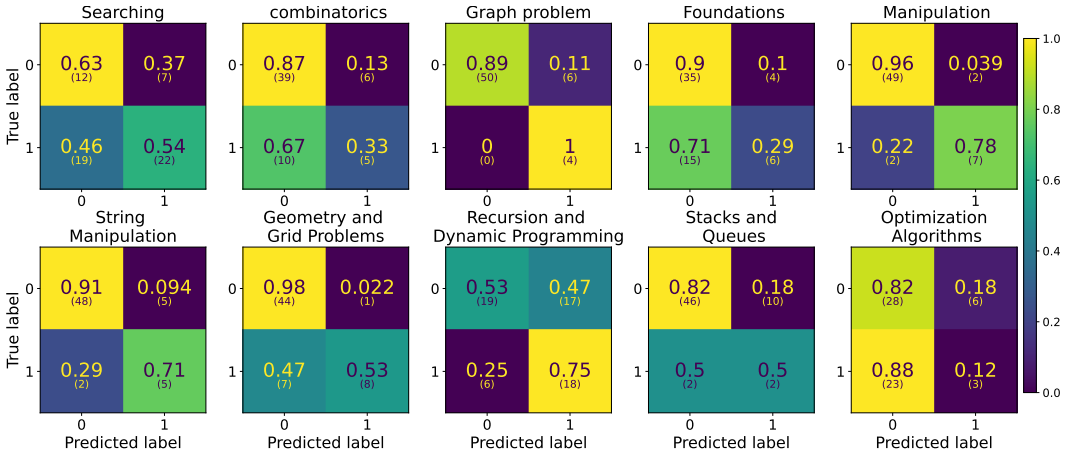


Figure 4.3: Faithfulness of semantic labeling. Confusion matrices for the multi-label classification task performed by the puzzle labeler. For each semantic descriptor, we report the confusion matrix where rows indicate the ground truth presence (1) or absence (0) of the skill while the column indicates its detection (1) or non-detection (0). We thus read from top left to bottom right: true positive, true negative, false positive, false negative rates (sample size in parenthesis).

Measuring diversity

Diversity in semantic space. Figure 4.4a to 4.4e report the evolution of various diversity measures as a function of the number of puzzle-solution pairs generated by the puzzle generator. Semantic algorithms (ACES and ELM semantic) better explore the semantic representation space: they discover more cell beyond the cells covered by P3's train set (4.4b), more cells in general (4.4a) and generate more puzzles beyond the cells covered by the train set (4.4d) and their cell distributions have higher entropy (4.4e). ELM algorithms generate more valid puzzles in general, but the non-semantic ELM mostly generates puzzles falling in cells covered by the train set (4.4c vs 4.4d). Our goal-directed ACES generates puzzles whose cell distribution has higher entropy than other baselines (4.4e). Algorithms that optimize for diversity in the semantic space were expected to achieve higher diversity in that space, but does it translate to higher diversity in other representation spaces?

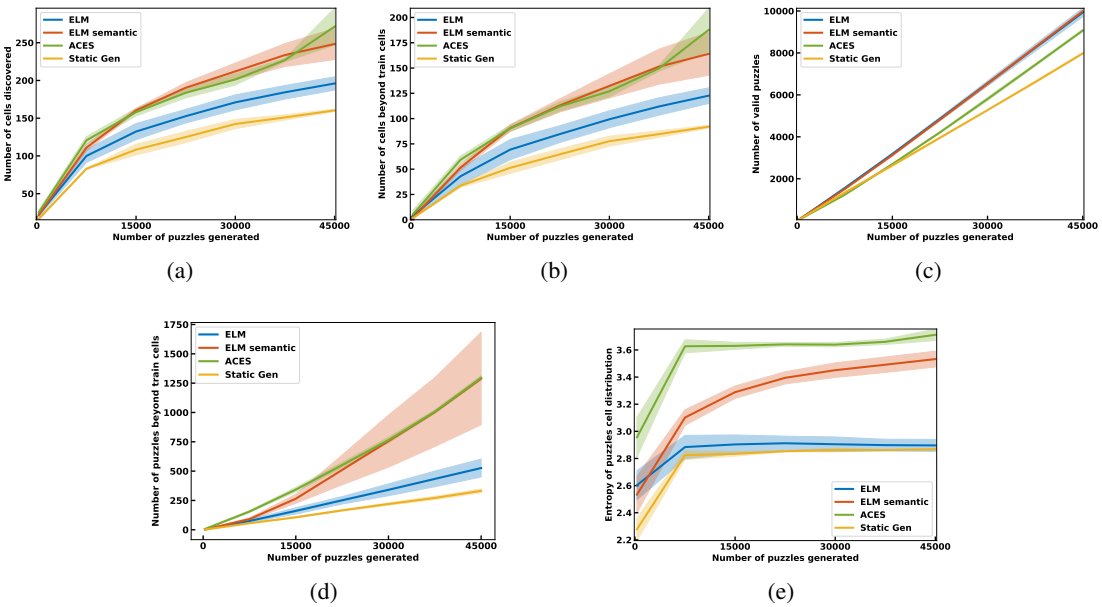


Figure 4.4: Diversity of generated puzzles in semantic space. We report the evolution of several diversity metrics computed in the semantic space as a function of the number of puzzle-solution pairs generated by the puzzle generator. Semantic algorithms (ACES and ELM semantic) achieve higher diversity in the semantic space.

Diversity in embedding spaces. Figure 4.5a to 4.5c report a diversity metric (Friedman & Dieng, 2022) computed over three different embedding spaces (see Section 4.2.4. ELM demonstrates relatively high diversity in the embedding space it uses for optimization (4.5a) but lower diversity on other embedding spaces (4.5b, 4.5c). ACES achieves the highest diversity in the two WizardCoder embedding spaces (4.5b, 4.5c) while ELM semantic reaches the highest diversity in the Code5P embedding space (4.5a). These results demonstrate that optimizing for diversity in our custom semantic space also leads to higher diversity in other representation spaces. Our autotelic ACES achieves significantly higher diversity overall.

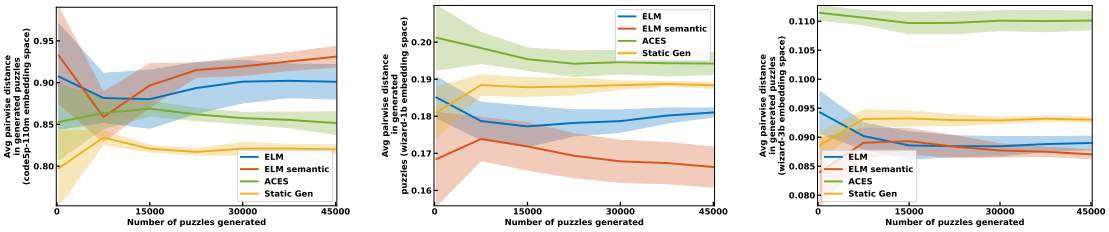


Figure 4.5: Diversity of generated puzzles in embedding spaces. We report the evolution of the pairwise distance between puzzle-solution pair embeddings as a function of the number of generated puzzle-solution pairs generated for three different embedding representation spaces (average across seeds).

Qualitative inspection of samples

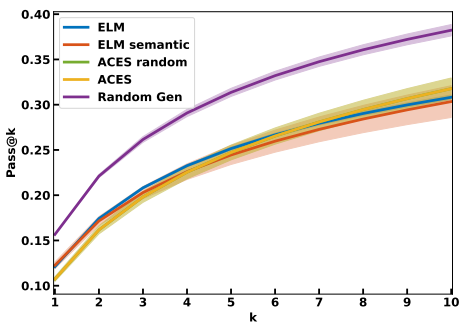
We here describe the most remarkable trends that we have observed by manually inspecting the data (see Appendix D.3). One tendency of the generation process across all experiments is to shift the definition of what a puzzle is. In the original formulation, the problem `f` implements a test that verifies a certain number of conditions are met, and `g` implements an *algorithm* that produces a value that satisfies the conditions. What the generation processes do in many cases is shift the algorithmic load from `g` to `f`, in which case `g` only provides arguments for `f`. We hypothesise this originally comes from the docstring description of puzzles in the seed examples, which are included in `f` but describe the behavior of `g`. Another important difference is what the puzzles look like: puzzles generated by the Static Gen baseline tend to be short and more math-heavy, while samples generated by ACES tend to be longer and more creative (see Appendix D.3 again for examples). Appendix Section D.4 provides additional visualizations such as 2D projections of the generated puzzles embeddings using UMAP (Appendix Figure D.5 to D.8), depictions of the evolution of the archives as a function of time (Appendix Figures D.4), as well as histograms for the distribution of skills and number of skills across generated puzzles for each of the algorithms (Appendix Figures D.1). The UMAP projections, in particular, give a good sense of the difference in distribution between methods.

Looking for performance predictors

Our contributions lead to larger diversities of interesting programming puzzles. Could this translate into higher performance on an arbitrary target distribution we might be interested in? We consider P3's testing set of puzzles to be our target distribution and measure the Pass@k performance of LLaMA 3B models finetuned on the generated sets of puzzle-solution pairs.

Figure 4.2.7 shows Pass@k metrics for various values of k. *Static Gen* performs higher despite having the lowest diversity of all algorithms on all considered metrics. Other algorithms spend more time exploring the full-extent of the space of programming puzzles (see Figures 4.4 and 4.5) and thus less time focusing on generating puzzles close to the training distribution. *Generating more puzzle-solution pairs and a higher diversity of them does not lead to higher performance in our setup.* Note that this can be perfectly fine. Our algorithms did not optimize for final performance on an arbitrary target distribution (here P3's test set) but optimize for pure diversity.

[17]r0.42



This said, we might be interested in figuring out how to constrain diversity-search to maximize performance on a target test distribution down the line. We thus test for correlations between the Pass@10 performance and both diversity metrics and metrics assessing the distance between

generated puzzle-solution pairs from P3's test set (target distribution coverage). Over the test metrics we only found: an anti-correlation with the number of valid puzzles generated, an anti-correlation with the number of cells discovered and an anti-correlation with the average pairwise distance between generated puzzles in the Code5P embedding space, all mostly driven by *Static Gen*'s low numbers and high performance. The FID score in embedding space (Heusel et al., 2017), number of puzzles in cells covered by the test set, number of test cells discovered or the average distance between test puzzles and their nearest neighbors in the generated sets computing in embedding space all measure how much the generated puzzle-solution pairs cover the test distribution of puzzles but none of them were found to be significantly correlated with the downstream performance metric.

4.2.8 Discussion

this contribution introduced ACES, an autotelic generative algorithm that optimizes for diversity in a custom semantic description space adapted for the generation of programming puzzles. ACES produces more diversity in semantic and several embedding spaces than the considered baselines, which results in the generation of interesting and creative puzzles as detected by manual inspection (Section 4.2.7).

Our last set of experiments uncovered a counter-intuitive result calling for more research: generating more puzzles of higher diversity does not translate into the higher downstream performance of an LLM finetuned on the generated data as measured over a target distribution of problems (here P3's test set). Surprisingly, we could not find any significant correlation between downstream performance and metrics measuring how much the generated set covers the target distribution. These results might be explained by more superficial drifts between the generated data and the test data: e.g. by the shift of the computational burden from the solution function to the problem function exposed in Section 4.2.7. This raises questions for future research: what are causal predictors of final downstream performance? Can we design the goal sampler of our autotelic diversity search to both search for maximal diversity while exploring the space of puzzle that will lead to good downstream performance? Can we characterize the trade-off between diversity and downstream performance?

Future work could also improve various aspects of ACES. One could replace the default uniform goal sampling with more sophisticated autotelic sampling methods (e.g. using novelty or learning progress intrinsic motivations, Colas et al., 2022c) or improve on the selection of in-context examples to help the puzzle generator. ACES currently explores a fixed set of semantic features and is thus somewhat constrained to *combinatorial* forms of creativity (Boden, 1998). Moving forward, we could give it the ability to come up with new semantic features or prune others as the exploratory process unfolds, opening the door to more *exploratory* and *transformational* forms of creativity or even *historical* creativity if this system were to interact with human cultural evolution processes, as defined in Boden's work on human and artificial creativity (Boden, 1998).

Pure diversity-search is useful beyond its data augmentation applications. On the first end, it could be used to generate diverse and interesting problems to educate the new generations of programmers. It could also be combined with autotelic solving systems where it would optimize for both interesting diversity and *quality*. High-quality problems could for instance be the ones that are *intermediately difficult* for the learner (Florensa et al., 2018), or those that maximize its learning progress (Oudeyer & Kaplan, 2007b). This paves the way for collaborative processes that endlessly co-evolve new interesting problems and their solutions, open-ended discovery systems that could be helpful for science (e.g. automatic theorem discovery and proving).

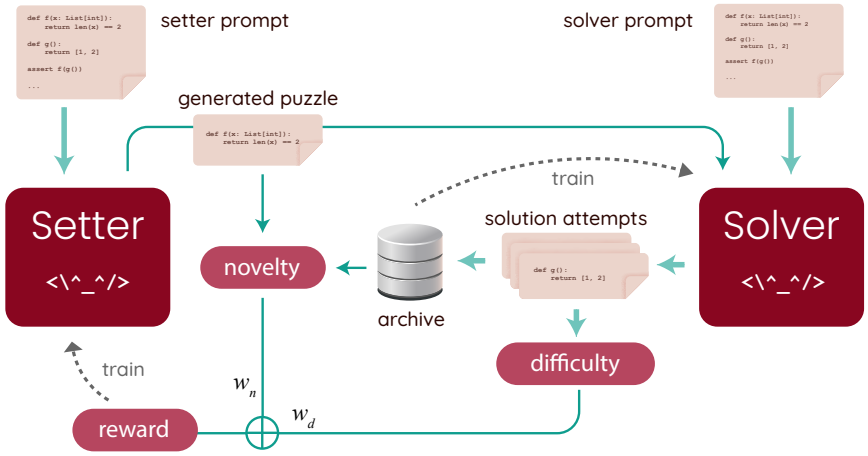


Figure 4.6: Overview of the proposed Codeplay algorithm. The Setter, a language model, takes in a few-shot prompt and emits a puzzle. This puzzle is given to the problem Solver who appends it to its few-shot prompt, and generates N_a candidate solutions. These problem-solution pairs are given to the Python interpreter, this gives us success or failure information on the solution. The number of successful attempts allows us to compute a difficulty reward. The generated puzzle is also compared with all previously generated puzzles to compute its novelty. Those two rewards are weighted and summed and used as the reward in a deep RL algorithm to train the Setter.

4.3 Contribution 6: Codeplay – Autotelic Learning through Collaborative Self-Play in Programming Environments (Perspective)

This section presents work in progress combining insights that have been gained from our contributions on linguistic agents in text environments as well as the work on ACES. The development phase of this project is still ongoing but we nevertheless have a few preliminary experimental results to present; we will thus present the approach and discuss it as a perspective work.

4.3.1 Introduction

We have see that ACES, our diversity-generating method in the space of programming puzzles, did manage do generate a higher diversity of puzzles according to several measures. However we are still missing a complementary quality metric for puzzles, that would signal how much they contribute to increases in coding skills of an agent. In this work we hypothesise that the high difficulty of a puzzle (while still being solvable), might be an appropriate metric (consistent with the theory of competence-based intrinsic motivation). We set ourselves in the Python Programming Puzzles domain (**?**), and we define an autotelic agent composed of 2 sub-agents: a goal setter that generates a puzzle, and a solver agent whose objective is to solve the generated puzzles. Both agents play a collaborative game where the setter has to create puzzles that push the solver to learn, and the solver sees and tries to solve puzzles in its zone of proximal development: neither impossible not too easy. First experiments highlight, in a restricted domain, the feasibility of this approach and the influence of the chosen reward function on the distribution of puzzles generated by the goal-setter.

Related work Closely related to this work are approaches to autotelic learning or automatic curriculum learning involving goal setter agents. (Florensa et al., 2018; Sukhbaatar et al., 2018; Campero et al., 2021; OpenAI et al., 2021; Mu et al., 2022). Also very closely related to this work are approaches for generating novel code puzzles for augmenting the capabilities of code-puzzle-solving language models (Haluptzok et al., 2022), as well as recent attempts to cast program synthesis as a reinforcement learning problem (Le et al., 2022; Yang et al., 2023).

4.3.2 Methods

We are interested, in this work, in open-ended problem and solution generation. The puzzles we use are again Python Programming Puzzles (P3) (Schuster et al., 2021), please see Section 4.1.1 for an introduction of the domain. We simply remind the reader that `f` refers to a programming puzzle and `g` refers to its solution. We've seen in the previous section that optimizing for diversity only did not necessarily lead to performance of the downstream solver. What might? In this work we hypothesise that optimizing generated puzzles for their difficulty, while still being solvable, could be an interesting direction.

Competence-based rewards

In this work we go back to the notion of *competence-based intrinsic motivations* (Oudeyer & Kaplan, 2007b) for linguistic agents. More specifically, we use the notion of intermediate competence to build an idea of interestingness. The intuition is the following: if we produce puzzles that are not too easy nor too hard, by finetuning on them a model will learn to perform tasks that it was previously unable to perform well. As in the work on autotelic agents in ScienceWorld, intermediate competence will be a proxy for learning opportunity. This is a valid assumption because the domain is supposed deterministic: solving a particular puzzle only depends on the generated solution and not on external factors in the environment. This prevents the agent from being captured by randomness, in an environment with no random libraries (which we can in principle filter out).

In the ACES work we generated both puzzles and solution from calls to an LLM, and passed them through the python interpreter to decide if the solution is a valid one. This prevents us from determining the difficulty of a puzzle. In this work we decouple the generation of the solution from the generation of problems: this allows to perform several trials of solution generation which improves the chance that a solution will be found, while giving us an empirical estimation of how hard the puzzle is for a given model. In this framework thus puzzles define *goals*, and solutions define *behavior*, in an open-ended environment defined by python programs. The goal space is thus as vast as the space of programs themselves, and include such simple tasks as simple string manipulation operations or additions, up to open problems in mathematics or programming.

What is the zone of proximal development for programming puzzles?To define our measure of competence-based intrinsic motivation we have to know how difficult a given puzzle is. In this work we are taking inspiration from the most widely used metric for evaluating code-generating models, the pass@k metric (Chen et al., 2021b). This reports the proportion of puzzles that have been solved by a model at least once in k attempts with a nonzero temperature. As expected if the probability of success is independent across trials, pass@k increases linearly on log scale as k increases (see the pass@k plots in Haluptzok et al. (2023) for an example). Concretely, this means that even hard

problems can generate signal for the solver if they are attempted many times. We define the difficulty D_θ of a puzzle p for a solver π^s_θ (with parameters θ) as the inverse probability of solving a p. This is the expected number of attempts before the solver succeeds, a metric intuitively linked to the pass@k. We build upon D to define intermediately difficult puzzles. An *adequately difficult* puzzle should not have a difficulty of 1 (because that means that the solver already knows how to solve the puzzle perfectly and would not benefit from training on it; we call a puzzle with $D = 1$ a *trivial* puzzle for solver π^s_θ); a puzzle should not be infinitely difficult (we call these *unsolvable* for the solver π^s_θ) because we cannot train the solver without a valid solution. We can of course not realistically know whether a puzzle is very hard or unsolvable. We thus in what follows set a maximum number of attempts N_a and consider all puzzles with $D_\theta(p) > N_a$ as practically unsolvable. All our preliminary experiments use $N_a = 32$.

Codeplay

Our insight in this work is that puzzle generation and solution generation happen in the same space, and can be performed symmetrically in a two-player game we call *Codeplay*. This collaborative game involves the interaction between a problem *setter* and a problem *solver*, both of which are language model-based agents. The goal of the problem setter is to generate problems which are of intermediate difficulty for the solver; the goal of the solver is to solve the problems that the setter generates. If this works well, the solver should improve on puzzles that are in its zone of proximal development, neither too difficult not too hard, and the setter should learn to stop generating puzzles that have become too simple for the solver. This asymmetric and collaborative game results, if this works well, in continuous generation of puzzles that are increasingly difficult for the base untrained solver, but that maintain a constant difficulty for the solver as it is trained. The training process results in the creation of a dataset of increasingly harder puzzles, their solutions, a setter model that has learned to generate difficult puzzles and a general solver model that knows how to solve puzzles similar to the ones generated from the beginning of training and to the end.

The collaborative nature of the game ensures, in theory, tamer dynamics than an adversarial one. In GANs (Goodfellow et al., 2014), the Discriminator and Generator are pitted against each other and because the tasks are asymmetric one of the player can win most of the time, which stops learning from occurring. Here, the goal of the Setter is to help the Solver learn: if the Solver's performance degrades the Setter should help it get back on track. In our description of Codeplay we present both training setups for the setter and the Solver, but note that our experimental results are performed by training the Setter only.

SetterWe instantiate our Setter as a pretrained language model, finetuned on the P3 domain. We cast the puzzle-generating problem as an MDP where the observation space is the list of all possible sequences of tokens, the action space is the list of all tokens and the reward is the intrinsic motivation measure we compute based on the difficulty of a puzzle. Transitioning from one (fully-observable) state to another is simply appending the emitted token to the observation: the environment is purely deterministic. This training setup is reminiscent of the one in reinforcement learning from human feedback (RLHF: Ouyang et al. (2022)), except in our case we do not use a reward function trained from human preferences but an intrinsic motivation metric based on the solver's abilities. Our Setter agent's stochastic policy is implemented by the pretrained language model's logits which are used to sample the tokens (temperature of 0.8 in our experiments). We use PPO (Schulman et al., 2017) as our training deep RL algorithm with an implementation based on the RL4LMs library (Ramamurthy

et al., 2022). As the value head in PPO we use a separate untrained MLP head on top of the language model backbone. In our preliminary experiments we use the small GPT-Neo-125M[1] pretrained on the Codex-generated dataset of (Haluptzok et al., 2023).

Setter difficulty reward To compute the intrinsic motivation metric used as the reward for the setter we use the notion of difficulty discussed above. We define the *difficulty reward* of a puzzle as a function ϕ of the difficulty $R^d_\theta(p) = \phi(D_\theta(p)) \in [-1, 1]$. The values of $f\phi$ are represented on the interval $[1, N_a]$; the reward is -1 for unsolvable as well as for invalid ones. ϕ is 0 for trivial puzzles, -1 for impossible of invalid puzzles, and smoothly interpolated between 0 and 1 on the interval $[1, 3]$.

Setter novelty reward In early experiments we noticed that while training the Setter with a fixed Solver, the Setter managed to generate intermediately difficult puzzles but collapsed to always generating the same output. To prevent this from happening we added a second *novelty* reward $R^n(p, \mathcal{D})$, with $\mathcal{D}$ being the archive of all previously generated puzzles. To compute $R^n(p)$, we encode p to an embedding e_p and compute the 3 closest neighbors of e_p in the archive in that embedding space. $R^n(p)$ is the average distance to these embeddings. Intuitively, if the model starts generating duplicate puzzles this reward will come close to 0. We note that this quantity naturally tends to decrease as generation unfolds because the density of puzzles in $\mathcal{D}$ gets larger. As a puzzle encoder, we use the hidden representation corresponding to the last token of the last layer of the non-finetuned Setter.

The total reward for the Setter is a weighted sum of the difficulty and novelty rewards:

$$R(p) = w_d\, R^d_\theta(p) + w_n\, R^n(p, \mathcal{D}) \tag{4.1}$$

SolverThe solver is slightly more straightforward to train. In principle there are two options. The first one is to train the solver through RL with a reward that depends on task completion. Since we need to generate solutions to compute the empirical difficulty of a puzzle, we might as well train the Solver at the same time, with PPO as well. The second option is to use behavioral cloning (BC) and simply finetune the Solver on successful puzzle-solution pairs. We favor the second option for two reasons: in preliminary experiments we have found this training process to be very unstable (presumably because samples in the PPO buffer are very correlated, since we repeatedly solve the same puzzles). And the second reason is that PPO is an on-policy algorithm, meaning that we cannot train on old samples and thus we cannot replay old puzzles to mitigate forgetting. Training language models with off-policy Deep RL methods is still an open field of study. In our preliminary experiments we only present the fixed-solver setup to demonstrate the effect of RL training on the Setter.

PromptsWe use a fixed few-shot prompt for both the solver and the setter. The few-shot prompt simply includes a series of puzzles and solutions from the tutorial set of the P3 training set (which are pretty short, so we can fit all of them). The puzzles and solutions are separated by `assert(f(g()))` statements. The prompt for the Setter finishes after an assertion, the prompt for the Solver (evaluated N_a times to produce the difficulty estimation) additionally includes the puzzle to be solved at the end. Compared with the ACES prompt, we do not include any instructions since the model was not

[1] https://github.com/EleutherAI/gpt-neo

```
from typing import List

def f(s: str):
    return "Hello␣" + s == "Hello␣world"

def g():
    return "world"

assert f(g())

def f(s: str):
    return "Hello␣" + s[::-1] == "Hello␣world"

def g():
    return "world"[::-1]

assert f(g())

...
```

Figure 4.7: Few-shot prompt used for the Setter (shortened for readability). In the Solver prompt, the puzzle to be solved is appended at the end.

instruction-tuned and is much smaller, so its understanding of English and general reasoning abilities are not as good.

Training details We train the Setter for 50 PPO iterations (Schulman et al., 2017). Each PPO epoch is composed of a data-gathering phase on 64 parallel environments, each environment being stepped for 128 steps (token generations). Once one of the parallel Setter agents has finished generating a puzzle p (generated a function `f` and output a double newline), we store p as a list of tokens in the PPO buffer and restart the episode. Once all the steps have been taken we perform 5 PPOs epoch on the collected buffer with the Adam optimizer and a learning rate of `1e-6`. We then discard the buffer and start again. We use a discount factor of 1, and generalized advantage estimation (Schulman et al., 2018) with $\lambda = 1$.

4.3.3 First experimental results

We present here a series of results studying Setter training. We perform several experiments to capture our ability to tradeoff puzzle difficulty and novelty. Specifically, we train the with the following weights (the names of the experiments are given in typewriter font):

- `Competence`: $w_d = 1$ and $w_n = 0$; Optimizing only for intermediate puzzle difficulty;
- `Competence_novelty_b` (balanced): $w_d = 1$ and $w_n = 1$; Optimizing both for intermediate difficulty and for puzzle novelty;
- `Competence_novelty_u` (unbalanced): $w_d = 0.5$ and $w_n = 1.5$; Same, but with higher emphasis on novelty;

- `Competence_novelty_m` (multiplicative): $R(p) = R_\theta^d(p) \times R^n(p, \mathcal{D})$;
- `Novelty`: $w_d = 0$ and $w_n = 1$; Only optimizing for novelty;
- `No_optim`: no optimization, the Setter is simply left to generate as many tokens as in the other experiments from its original distribution;

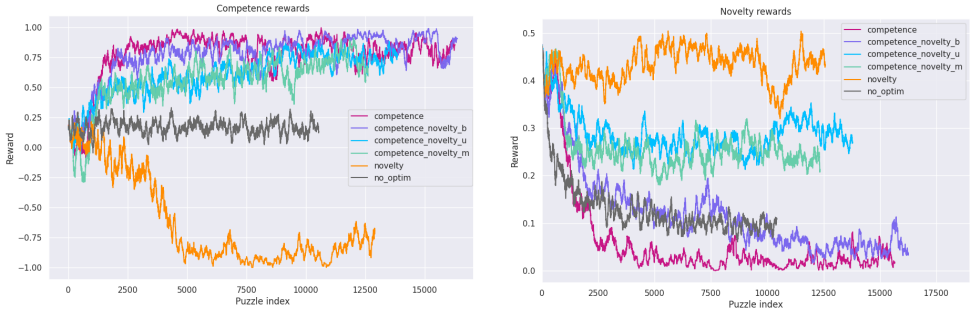


Figure 4.8: Competence and novelty rewards for different experiments. setter_competence has competence rewards only, setter_novelty novelty rewards only. setter_competence_novelty_b (balanced) sums both rewards with weight 1, setter_competence_novelty_u (unbalanced) sums competence with weight 0.5 and novelty with weight 1.5, setter_no_optim simply generates puzzles for the same amount of tokens and allows us to follow the natural decrease in novelty rewards, and setter_competence_novelty_m (multiplicative) multiplies both rewards with weight 1. All curves show smoothed averages over 100 puzzles. Curves stop at different points because the experiments perform are run for a constant number of tokens and the average lengths of puzzles are different across experiments.

We report the learning curves for both reward components R_d and R_n in Figure 4.8. We see that optimizing with non-zero weight for intermediate competence yields puzzles that are indeed intermediately difficult, compared to the original Setter which tends to create trivial puzzles (visible with the scores in the gray `no_optim` curve). The sample efficiency is best when w_d is high. Optimizing for novelty only leads to the difficulty rewards collapsing to -1 on average: on inspection this is due to the fact that the puzzles are becoming invalid, since invalid puzzles are not penalized when $w_d = 0$. When looking at the values throughout training for R_n, we see that the `competence` run, with no novelty reward, collapses close to 0 fast. Examining the generated puzzles for this configuration, we see that there are only a few puzzles generated repeatedly. Looking at the `no_optim` curve, we confirm that R_n naturally decreases for a constant distribution because of the increased density of puzzles of $\mathcal{D}$ in the embedding space. This serves as our baseline. The `competence_novelty_b` with equal weights on difficulty and novelty contains the collapse in diversity of generated puzzles as training proceeds to the same level as the `no_optim`. The decrease is less severe in the `competence_novelty_u` and `competence_novelty_m` cases, suggesting our ability to steer the model towards producing novel outputs with RL on this nonstationary reward. Note that for those two setting we do not lose the ability to optimize for intermediately difficult puzzles. And finally, the `novelty` run manages to keep the novelty metric high, but we've seen that this comes at the expense of the validity of the puzzles.

4.3.4 Further work and discussion

In this perspective work, we have cast the autotelic learning problem in the space of Python Programming Puzzles as a collaborative game between two agents: a Solver learning to solve python puzzles and a Setter learning to output puzzles that are intermediately difficult for the Solver. In first experiments examining the behavior of the Setter, we have seen in these early experiments that we are able to train language models with deep RL to optimize classical measures of intrinsic motivation such as intermediate difficulty or novelty of a sample. This allows us to combine the paradigm of autotelic agents trained with deep RL (Colas et al., 2021) with the flexibility, generality, and massively multitask nature of pretrained language models.

In Section 4.2 presenting ACES, in the discussion we concluded that optimizing for interesting diversity alone was not sufficient for producing a dataset that could be used by a downstream model to increase its performance. We care about performance because building increasingly capable models is one of the drivers of open-endedness. An open-ended agent must learn from its experience and generate harder and harder problems for itself by using this acquired knowledge. Insofar as generating intermediately difficult puzzles helps a model improve, our Codeplay experiments show that the tradeoff between generating diverse puzzles and intermediately difficult puzzles can be negotiated by integrating different components in the reward function.

Of course, these results need to be extended by training the Solver as well, which we have actually found more difficult than training the Setter. The setting needs to be scaled as well, because the diversity and quality of the generated samples improve radically as the model capacity and training are scaled up. There are two final objectives for this work: this first being generating a sequence of puzzles that are increasingly difficult for the fixed, untrained Solver but that maintain a constant difficulty for the Solver as it is trained; the second objective would be to find a way to steer the process towards the generation of artifacts that are interesting to humans. If these two conditions are fulfilled we will have demonstrated the embryo of a truly open-ended linguistic agent, steerable towards problems that humans find interesting.

4.4 Chapter discussion

Let us recapitulate the contributions of this chapter. We have motivated the implementation of autotelic agents in programming domains such as writing Python code. Programming defines a truly open-ended space to explore and is not constrained to the effort put into environment design; it thus seems like an excellent playground for learning to define one's problems. Code execution provides a ground truth against which to test task completion, allowing us to sidestep the need for a language model-based reward function such as the one in LMA3. We thus set ourselves in the Python Programming Puzzles (Schuster et al., 2021) domain, composed of puzzles of varying difficulty and their solutions.

In our first contribution we have defined the notion of semantic descriptors to help us design a space within which to measure *interesting diversity*. These semantic descriptors, in our programming puzzle space, are defined as a combination of categories of programming problems, we call a combination of *skills*, and they are labelled by an LLM given a particular puzzle. We build upon these semantic descriptors to propose ACES, an autotelic algorithm for diversity generation based on sampling goals in skill space and prompting the LLM to generate puzzles with that label, conditioned on few-shot examples sampled in the neighbouring cells of the targeted one. We showed that ACES,

closely followed by a variant of Evolution Through Large Models (ELM: Lehman et al. (2022)) using the semantic descriptors as well, performed higher compared to baselines on several measures of diversity. We concluded by discussing that further work should try to measure the correlates of performance of a downstream model, finetuned on a generated dataset, on a given test dataset: is there a tradeoff between the diversity of generated data and downstream performance? We have also discussed the significance of the work for implementing open-ended linguistic agents and discussed the link between ACES and different notions of creativity (Boden, 1998): *combinatorial* creativity for generating puzzles and solutions that mix a novel skill combination, and *exploratory* or *transformational* creativity if the system is allowed to define its own categories and add them on-the-fly to the semantic representation. This is an intriguing research direction since it is unclear how LLMs would fare at generating completely novel concepts.

In our perspective contribution we have taken a complementary view on autotelic learning in the P3 domain. We have introduced Codeplay, an algorithm framing autotelic learning as a two-player game where a Setter generates programming puzzles for a Solver that tries to generate a solution. Both Setter and Solver are language models. We have performed some experiments with a fixed Solver, using RL to finetune the Setter using a reward composed of an intermediate difficulty motivation as well as a novelty motivation. We have discovered that optimizing for only one of these motivations leads to mode collapse and generation of invalid puzzles respectively, but that by balancing both rewards one can generate a dataset of intermediately difficult and diverse data. We have then shortly outlined that we should complete the loop by training the solver, as well as implement ways to steer the generation of puzzles in a direction (for instance toward making progress on a particular pool of puzzles).

These two works are very synergistic: once progress has been made on Codeplay one could combine it with a prompt constructed with an autotelic algorithm from a semantic descriptor space such as ACES. Interestingly, our notion of goal (and thus the kind of autotelic learning taking place) is different in the two works. In ACES, a goal is a 10-dimensional semantic vector in skill space, and a realization of this goal is a puzzle and its solution that falls into this category; in Codeplay, a goal is a puzzle and its realization is a solution. This means that combining both approaches could lead to a form of meta-goal: when generating a puzzle for itself, the agent first selects a goal in this meta-goal space and uses it to condition the generation of a puzzle that is given to the solver. This would extend the intrinsic motivations for the solver with a notion of goal-satisfaction at a higher, more abstract level.

If we now recall the discussion of Chapter 2.3 we had identified a number of technical and conceptual limitations of our approaches, either the study of agents in ScienceWorld or our LMA3 agent. One limitation was the difficulty of learning a truly multitask agent: we solve this by leveraging pretrained language models, large and small, as our policies, and finetuning on diverse enough data. The second limitation we listed was the lack of sync between the goal generation and the abilities of the policy at a certain point in time. We solve this issue by either sampling neighboring skills in the semantic archive in ACES or by optimizing the Setter with a reward based on the current Solver's competence. The final limitation we listed was the intrinsic limitation of the text-based environments we used: we have dissolved this issue by setting ourselves in the open-ended space of python programs.

Autotelic agents operating in the space of programming puzzles opens up numerous possibilities. First, since we can interact with many simulated and real-world domains through APIs, we do not necessarily lose in generality by restricting ourselves to code inputs and outputs. The most striking

recent examples of what can be achieved with code agents in a nominally sensorimotor environment is the Voyager agent of Wang et al. (2023a) (which is concurrent work to LMA3 and is conceptually similar). This Minecraft agent interacts with its environment through writing code that implements given behavioral programs based on some predetermined primitives. Second, programming itself beyond solving puzzles is a very wide domains, and we could imagine autotelic developer agents coming up with creative ideas for entire apps and learning by doing; their performance might be boosted by clever prompting techniques and self-evaluation (Shinn et al., 2023; Yao et al., 2023) and corrections from error stack traces, which we, for now, do not consider in our solver agents.

Next to general-purpose programming and development, settings such as these could be applied in conjunction to formal method systems for automated proof generation (Moura & Ullrich, 2021) to implement automated exploration algorithms for mathematical theorems and their proofs. Such systems could invent new theorems (understood here as goals) and try to achieve them (write the proof in the formal language of interest). These agents could maintain a repertoire of theorems previously proved and use these in inventing harder and harder theorems, guided by some intrinsic motivation. Setups where reuse of previous goals is possible are also interesting because they map quite clearly to the notion of stepping stone of Stanley & Lehman (2015). Additionally, this opens up a new class of intrinsic motivations: intuitively, a theorem is interesting if it can be reused in many other theorems. Another idea, related to the notion of *evolvability*: a theorem is interesting if it can be mutated into a large number of novel theorems (in something like ACES, this could mean incrementing the interestingness of puzzles appearing in a prompt that generated novel puzzles in the semantic space). Programming or theorem-proving autotelic agents retrieving skills from a library could thus be a generative setup for investigating novel or under-investigated forms of intrinsic motivation. This research could have important applications for mathematicians: autotelic theorem-generating and proving agents could be used as research assistants discovering new theorems and proofs alongside them.

Chapter 5

Conclusion

5.1 Summary of contributions

We are now ready to conclude, let us summarize, for each chapter, our contributions and what we have learned.

5.1.1 Autotelic agents in text-based worlds

In our first contribution of this chapter, we have studied an autotelic agent based on deep RL in a complex text-based environment called ScienceWorld. ScienceWorld features rudimentary physics, biology and thermodynamics, and we have build a hierarchy of nested language goals this agent could master. The agent learns about goals through hindsight: it performs exploration of the environment and a social partner relabels its trajectory if one of the relevant goals has been achieved. In that study we have determined that sampling goals (at the beginning of the episode as well as within the replay buffer) according to their intermediate difficulty was beneficial for learning, recovering one of the simplest forms of competence-based intrinsic motivation. Additionally, the social partner needed to restrain its relabeling to a small enough number of goals so that the goal-conditioned RL policy could learn. In a sense, this limitation is a limitation on the multitask abilities of the agent: with a higher sample efficiency the social partner could give hindsight feedback on many trivial goals that would be mastered without perturbing the agent on its way to mastery of the harder ones. While the goals of the agent were based on language, which allowed us to leverage abstraction properties, and while some of our agents used an intrinsic motivation measure to sample their goals, there was no creative use of language.

In our second contribution, we have moved on to consider the problem of creativity in language-based autotelic agents, namely, how could those agents invent an open-ended sequence of goals? We have leveraged LLMs as models commonsense reasoning to creatively invent and manipulate new goals: we called this technique LMA3 (Language Model Augmented Autotelic Agents). We have set ourselves in another text-based environment called CookingWorld, and have used the LLM as a tool both for generating new goals, relabelling trajectories with lists of goals effectively accomplished, and as a general reward function allowing us to measure the completion of arbitrary goals. The policy itself was simple, based on a mapping of skills to sequences of actions. This allowed the agent to master a wide collection of skills, approximately one per episode. We studied the effect of variations in the prompt and found that adding chain-of-thought as a reasoning technique, as well as suggestions and tips on what makes a goal interesting, increased dramatically the amount of generated goals,

giving a first hint on how to combine open-ended goal generation with biases towards interestingness. We also noted that evaluating these agents proved challenging: how should we balance the diversity of goals with their depth for instance, so that we can reward both generalist and specialist agents? We also ended on a note calling for extensions to richer, truly open environments: LMA3 could invent crazy and creative goals but will never achieve it in its limited household environment.

5.1.2 Object representation for language grounding

In our third contribution third contribution, motivated by making progress on flexible reward functions for autotelic agents, we have decided to investigate good architectural designs for neural networks working with collections of objects. We have defined the SpatialSim to test the abilities of different neural networks to compare spatial configurations of objects. SpatialSim is a testbed composed of a configuration identification task and a configuration comparison task. We test a battery of neural networks based on Graph Neural Networks and identify the design that is most able to learn those two tasks: namely, the Message-Passing Graph Neural Network (MPGNN) that models relations between object learns better and is the most robust.

In out fourth contribution we build upon the intuition that language grounding might be easier within an object-centric framework that includes relational inductive biases (Battaglia et al., 2018a) and on the fact that Transformers, like MPGNNs, are architectures explicitly modelling the relation between sets of objects, to propose a study of object-based language grounding. Our overall aim was to propose grounding modules for language that incorporate spatial relations to refer to objects as well as temporally extended actions and the past tense. We thus defined a spatio-temporal language domain based on an artificial language that nevertheless incorporates spatial and temporal relations. In this setting we have define several Transformer architectures all incorporating different inductive biases with respect to how to process trajectory information and how to relate it to a language goal. We have discovered that for this task, maintaining object permanence, e.g. integrating information with respect to time first, was a reasonable baseline, but that using a flat architecture where all elements of the trajectory could interact in an unstructured way was the best.

We concluded the chapter with a pessimistic note. While it is true that language grounding can be achieved using multimodal Transformer-based models and that much progress has been made on this front in recent years, we believe that more progress will be made in autotelic learning by studying agents in language-based worlds directly. This way, we solve the issue of generalization of these models to novel experiences and we sidestep the difficult problem of low-level motor embodiment. Autotelic agents need not live in the same niche as we do.

5.1.3 Learning to generate programming problems

In our fifth and ixth] contributions, we have begun addressing the issue of the limited nature of the text-based environments we used in our first and second contributions. Our insight is that incarnating autotelic agents in a programming domain ensures, through the unlimited expressiveness of modern programming languages, that the agent's possibilities are never limited by its environment. Autotelic agents operating with programming goals can check the results of the execution of the code, simplifying the issue of implementing goal-satisfaction functions for arbitrary language goals, something that we achieved with a large language model in our second contribution. We select a specific expressive programming domain which is one defining Python Programming Puzzles (Schuster et al., 2021) (P3)

which is broad and expressive: puzzles can range from the simplest list manipulation operations to open problems in mathematics.

In our fifth contribution, coming back to our aim of problem-generating systems, we have studied the generation of an interesting diversity of programming puzzles. Taking inspiration from quality-diversity (Pugh et al., 2016) research and population-based autotelic algorithms (Baranes & Oudeyer, 2013c), in this work we aim to generate puzzles that maximally cover an archive composed of cells. Individuals belonging to different cells should differ in meaningful ways. To define this space of cells that should capture dimensions of variation that are useful for us, we turn to LLMs again and define a semantic representation space as a combination of programming categories. A puzzle's representation is a binary vector that indicates whether the puzzle is part of a given category, as judged by the LLM. ACES (Autotelic Code Exploration based on Semantic descriptors) targets goals in this semantic space, instructs the LLM to target it and displays some few-shot examples from the cells closest to the one being targeted. The generated puzzle and its solution are generated and kept only if the solution solves the problem. This performs better than baselines on several measures of diversity, although this does not immediately translate to better downstream performance when finetuning a smaller LLM on generated data and testing its puzzle-solving performance on the P3 test set. This opens the question of finding correlates, for datapoints and whole datasets, of downstream performance. This is important for having coding agents that are effectively getting better at solving puzzles after training on their own data. Another promising avenue for work is updating the semantic representation on the fly with novel concepts, which could help us model other forms of creativity than compositional (Boden, 1998).

In our sixth and final contribution (preliminary work), we take a complementary view on autotelic agents generating their own puzzles and solutions. Whereas in the previous contribution, goals were targets in the semantic space, and puzzles and solutions were generated in one go, in this work we consider puzzles themselves to be the goals, and the solutions to be the skills required to solve them. We define the autotelic learning process as a collaborative, asymmetric game between two agents. In this game we call Codeplay, the Setter is trying to generate diverse and intermediately difficult puzzles, and the Solver is trying to solve them. Both networks are language models trained on code, and using deep RL on top of language models allows us to piggyback on top of language models' strong generalization and multitasking abilities while optimizing arbitrary rewards. Preliminary results show our ability to train the Setter LM with RL, and to successfully navigate between generating novel puzzles (that might be syntactically invalid) and intermediately difficult one (that might be repetitive). Extending the results to include training of the Solver will be needed. Codeplay is reminescent of GANs (Goodfellow et al., 2014), as well as other approaches in intrinsically motivated learning (Sukhbaatar et al., 2018; Campero et al., 2021), and open-endedness (Wang et al., 2019). This perspective work brings together the ideas around co-evolution and self-play, using pretrained language models as policies, in a programming domain. It is moreover very synergistic with ACES, since the prompt generation techniques of ACES could be applied to Codeplay to generate even more diverse puzzles (and solutions!).

5.2 Discussion

The work in this thesis opens many avenues for further work, many persepectives. After our short recap of our contributions, let us speculate and explore some of these ideas in more detail.

5.2.1 Open-endedness and large models

We are not the first to notice the great synergy between large, flexible language models and open-endedness research. The capabilities of such models mean, as we have remarked elsewhere, that they can serve as a basis to build truly massively multitask agents, mitigating issues of lack of generality in deep RL goal-conditioned policies trained from scratch (see Section 2.2): we use this property to build our Codeplay Setter. These language models can learn in-context, making them a flexible backbone to any kind of exploration algorithms in many different domains, as was demonstrated with LMA3 and ACES. But even more importantly, LLMs contain all kinds of implicit knowledge about the world, and about the preferences of many different groups of humans.

As was already argued by Colas et al. (2022a) and Zhang et al. (2023), a particularly interesting application of LLMs for open-endedness research is to serve as Models of Interestingness (MoIs), encoding what humans find relevant and salient about a particular situation. This means, instead of using some form of knowledge-based or competence-based intrinsic motivation as a reward or fitness for particular goals experienced by an autotelic agent, the interestingness of a goal would be computed based on the response of an LLM. We rely on this mechanism when asking the LLM-based modules of LMA3 to generate or relabel goals that are interesting (and we remember that explicitly asking the LLM to consider goals that are relevant to humans, and providing it some examples, greatly expands the amount of goals generated by the agent). The Voyager agent of Wang et al. (2023a) uses a similar principle, using GPT-4's knowledge of the Minecraft game to set relevant goals within the particular context of an agent. The idea of LLMs as models of interestingness was also implemented in the work of Zhang et al. (2023) and Bradley et al. (2023a), respectively to filter the goals an agent could target and to serve as a quality measure in a quality-diversity algorithm from AI feedback. In all those works, the LLM is considered as a MoI in general, but Colas et al. (2022a) goes further, and discusses the idea that this MoI can be contextual. Different people find different things interesting, and as such there might not only be one consistent and definitive measure of interestingness encoded by a language model. Since language models are superpositions of perspectives (Kovač et al., 2023; Andreas, 2022), Colas et al. (2022a) argue that we could use LLMs as a model of interestingness from a particular perspective, leading to a diversity of possible exploration dynamics.

The idea of LLM MoIs calls for the investigation of LLMs' abilities to generalize what humans find interesting in novel domains. If these MoIs are used in a completely open-ended architecture, they might be asked to judge the interestingness of goals or skills on which we have no training data. Additionally, in human cultural evolution (or history of science), paradigm shifts and revolutionary ideas were had by highly motivated and persevering individuals and small groups going against the conventions and accepted norms of the time they were in. Presumably, these people were motivated by other things than social convention and culturally refined taste for problems. These interests might be additionally very personal, tied to the identity of the individual or group of individuals. Sometime in the future, when research on MoIs matures, these questions might need to be addressed.

LLM MoIs also raise an interesting question in the study of intrinsic motivation (IM) as a field. This question has the structure of a bitter lesson (as first defined by Sutton (2019)). It might be the case (and plausible) that using IMs based on LLMs is more efficient than relying on more traditional models of IMs like the ones presented in Section 1.2.3. An important part of the study of intrinsically-motivated and open-ended learning was historically finding models of IM, perhaps through the study of young children or of intrinsically-motivated behavior in adults, and validating this insight by proving that this IM worked well as a reward for an intrinsically-motivated agent. This approach, of studying a particular phenomenon to gain an understanding of it and implementing that knowledge in

a machine to build AI systems is precisely what Sutton deems to have failed in comparison to generic methods that scale with compute. This means that in many domains of AI/cognitive science, the search for scientific truth and descriptive, parsimonious models of the mind has become decorrelated from the quest to build better systems (while in physics for instance deep understanding and engineering have always gone hand in hand). Section 2.1.1 discusses this split in the language domain, between linguistics and NLP. Coming to terms with the bitter lesson in intrinsically-motivated and open-ended learning might mean that our quest to build open-ended autotelic machines might become unrelated to any form of scientific understanding of intrinsic motivation in humans. It is still too soon to decide on this question, but we find it a compelling potential lesson to contemplate.

Another thread at the intersection of LLMs and intrinsic motivation research is the two approaches to integrating large models and autotelic algorithms. These two approaches are represented in this thesis, and we could summarize them as such: the first perspective is using LLMs as tools for autotelic learning, and the second perspective is using autotelic learning to expand LLM's capabilities and allowing them to learn about new domains (and possibly invent their own). These two perspectives can of course be complementary, especially in the longer term, but in experimental practice they entail very different setups because they mean, for now, working with models of different scales (although this is bound to change with the trend towards smaller and overtrained language models (Hoffmann et al., 2022; Touvron et al., 2023b)[1]). The first perspective is perhaps best exemplified with the LMA3 method: we use an LLM as an external black-box to invent goals, compute rewards and relabel trajectories. The second perspective is best exemplified by the Codeplay perspective: we use autotelic learning as a two-player game to expand, in our experiments, the distributions of puzzles invented by the Setter. ACES is an example of both, through the idea of distillation: using a larger LM to explore the space of problems to teach a smaller model about a novel domain. The distillation perspective makes it, in our mind, less compelling: this is because it implies that you always need a larger model whose abilities stay fixed, to allow a model to go beyond its training data. It is more impressive to get an LLM to expand its own skill repertoire by exploration without too much scaffolding. We believe that autotelic learning is a promising avenue to finetune language models to expand their capabilities, and that language models are the perfect backbone for multitask policies, goal generation and other flexible modules. Future research at the intersection of these two domains seems particularly interesting to us.

One of the compelling perspectives that emerges at the intersection of autotelic AI and LMs is that of data augmentation, through self-generation of problems and their solutions. ACES takes a first step in this direction by focusing on the diversity of generated data, but an important followup should be finding ways to compute correlates of the programming abilities of a language model, specifically computing an indicator of the amount of programming ability that a particular problem confers as training data. If we had access to such a metric, we could generate data that is both diverse and that pushes the boundaries of competence of a downstream language model (this amounts to a mixture of novelty-based and competence-based intrinsic motivation); this metric could also be used as a reward in an RL setup. The wider perspective is interesting: as we noted in our introduction (Section 1.1.5), data, not number of parameters, seems to be the bottleneck in training more capable models: having a principled way to generate high-quality data through a measure of the competence this data confers would be significant. Autotelic learning would also allow us to generate high-quality data in domains where there is none, or very few.

[1] See also the recent release of a small overtrained model at: https://mistral.ai/news/announcing-mistral-7b/

5.2.2 Languages, problems, math

Another discussion thread around the research presented in this thesis is the suitability of language to express problems. We have worked within text-based games domains which mimic very simple versions of household environments with rooms, objects, and in the case of ScienceWorld some physics, thermodynamics, and biology. We've seen that these environments are quite limited. One reason is that modelling the complexity of the real world would mean implementing a complex game engine (and then rendering the simulation in language, as in the text version of Minecraft (Wang et al., 2023a)). Another limitation is that mastering complex skills in such a toy environment is less useful than mastering skills we humans practice in language every day, such as writing stories, summaries, etc. We've then moved on to the space of Python programs, which does not have the limitations outlined above and additionally allows us to evaluate our agents without relying on a learned reward function. In general, training code-generating autotelic agents could lead to even more general skills, since these agents could also interact with elements of the real world through APIs (even if care should be exercised in this domain).

A perspective opened by using programming as an exploration domain is the explicit hierarchy this entails through function reuse. Python puzzles might be too specific for this to work well, but more general programming and development (such as in the case of an app-developing autotelic agent) could store written functions (mastered skills) and retrieve them as needed. We could even compute other intrinsic motivations based on the amount of re-use a particular function has, based on the intuition that something interesting often means something useful.

Another related domain for autotelic exploration is formal methods, especially for automated theorem proving. Based on the proof-program isomorphism, formal languages for expressing domains of mathematics have been developed these past few decades in the hope that one day all mathematicians could implement their theorems and proofs directly in these languages. The proofs could then be guaranteed by the system without requiring manual, time-consuming inspection of the details by specialists. While these formal frameworks have been useful for deriving the proof of some historically significant theorems, they have not been widely adopted by mathematicians due to the tediousness of proving even the simplest of arithmetic lemmas. We believe that automated theorem proving is a perfect setup for autotelic exploration. In this context, theorems are the goals of an autotelic agent, and proofs are the behavior to reach these goals. As in the case of programming, there exist skill hierarchies (in the form of libraries of theorems that can be imported as needed for a proof). Selecting the goal to prove requires a measure of intrinsic motivation over theorems (which is a fascinating avenue to explore because we might benefit from insights into what mathematicians find beautiful or interesting). Exploration of mathematical identities is a pure, model version of exploration of scientific truths more generally, except that when doing math one only needs to interact with the rules of logic and not perform costly experiments in the real world. Mathematical autotelic agents could also define their own mathematical objects (through type definitions), a model of concept creation (and of exploratory creativity (Boden, 1998)). And finally, autotelic theorem-proving agents could be useful to real mathematicians: they could serve as assistants to discover new theorems and lemmas. They could help fulfill the original dream of the creators of formal methods for automated theorem proving: the tediousness of writing proofs in a formal language is not really an obstacle. We could go further and argue that any ML application for finding or proving theorems should use a formal language, because no mathematician will be bothered to check the validity of free-form proofs drafted by a language model (such as in Lightman et al. (2023)). Technically, autotelic theorem-proving agents could rely on a mixture of language models (GPT-3.5 and GPT-4 have knowledge

of Coq and lean and can prove basic identities one-shot), advanced prompting techniques (like for instance Yao et al. (2023)), feedback from the compiler, and reinforcement learning. One can also incorporate elements of state-of-the art ML for automated theorem proving techniques (Lample et al., 2022).

Building automated mathematicians (we are reminded of Lenat's pioneering work here (Lenat, 1976, 1983)) will also require tuning interestingness functions (possibly also based on large, learned models of interestingness) to the interest of individual mathematicians. If you imagine an automated mathematician as a research assistant for a flesh-and-bones one, the latter needs to be able to distill their specific interests to guide the machine's explorations, rather in the spirit of the meta-diversity work of Etcheverry et al. (2020) building a hierarchy of spaces to measure diversity in, that can be influenced by a human's choices and interests.

5.2.3 The autotelic game

Another thread emerging from this thesis is the idea of autotelic learning as a game between two parties. The idea is not novel, and has been explored in work on adversarial goal generation (Florensa et al., 2018; Sukhbaatar et al., 2018; Campero et al., 2021) and in the powerplay framework (Schmidhuber, 2011); but it gains renewed importance as we start considering domains where generating good problems is as hard as finding good solutions. This has been recognized by researchers trying to explain the principles of good research: we are reminded of Hamming's speech on what makes good and effective research. He emphasizes regularly asking oneself what the most important problems in one's field are. Taste in research problems comes slowly and through repeated practice, and through deliberately reflecting about which research threads thrive and which ones die out, what ideas are important to the field, and so on. Michael Nielsen, in his essay "Principles of effective research", is even more explicit. He defines two ideal types of researchers (in theoretical physics): the problem-solver and the problem-creator. The problem-solver scours the literature for well-posed, important (and difficult!) problems to solve, and uses mostly existing tools to solve them. The problem-creator's work is more creative: they connect different domains, or look for abstract patterns, to create novel lines of inquiry. In effect, each researcher is a mixture of both types to a differing degree, and a field, of course, advances through the interaction of these two types of research. We are thus well-advised to model autotelic learning as an interaction between these two types of agents, in a two-player (or multi-player) game. These two agents could share part of the same weights, because presumably representations useful for generating problems might be similar to representations useful to solve those problems. This makes linguistic autotelic learning akin to a type of self-play, which has demonstrated its ability to go beyond human competence in games (Silver et al., 2017).

This leads us to consider the multi-agent perspective on autotelic learning: researchers creating and solving problems do not work in isolation but are connected in social networks (see Nisioti et al. (2022) for a discussion of the impact of connectivity in societies of agents making discoveries). Problem-creating agents and problem-solving agents, perhaps each in their own specialty and each with their own interestingness function, could interact in a society not unlike an asymmetric, linguistic version of AlphaStar (the StarCraft II-playing system, see discussion in Introduction, Section 1.1.4, where open-ended dynamics of strategies and counter-strategies emerged from multi-agent interactions between agents that had slightly different objectives. Remember that AlphaStar agents had a first pretraining phase (behavior cloning from human trajectories), that in our case is akin to the pretraining phase of language models. Multi-agent RL then serves to explore beyond this dataset, which is exactly

what we want in an autotelic, linguistic, open-ended system. For societies of autotelic agents (or humans and agents) to work well, it might additionally be interesting to consider the lesson of Tomasello (2021): one of the reasons for the success of the human species is the ability to form join goals with collaborators at a small scale, and at a larger scale to conform to society-wide norms and expectations as well as striving towards institution-level goals (such as when working for a government, large company, and the like).

5.2.4 Ethical considerations

We want to conclude this thesis with a short discussion on the ethics and potential risks of autotelic AI. This discussion could warrant a whole essay, and we will only touch a few points in what follows. As AI has matured as a research field and has led to tangible, revolutionary applications, concerns have risen in the community and beyond about the potential risks and downsides of this technology. Two distinct (sometimes opposing) communities have formed around the discussion of the risks of AI, communities that we will term AI Ethics and AI Alignment.

The former worries about the potential downsides of AI for minorities that are marginalized, about biases in algorithms and the entrenching of systematic injustice: the community has ties to movements linked with social justice, and to worries about algorithmic decision-making already contributing to the strengthening of biases before the successes of large-scale language models, especially in school systems, in the allocation of loans, and in law enforcement (O'Neil, 2017). Another related worry is that automation brought about by AI will massively displace jobs and increase inequality, perhaps contributing to the rapidly increasing wealth of the companies (and boards of investors) deploying large-scale models and the coming obsolescence of white-collar workers which will be forced to adapt or be rendered useless in the workforce.

The AI alignment field worries about a set of problems that are less tangible and that come from theoretical discussions about the risks of AI that come from a perspective inspired by science-fiction works: that a future superintelligent AI's goals will not be aligned with ours and that this will result in large risks (for some authors, existential) for humanity as a whole (Bostrom, 2014a). Alignment refers to the process of making AI's goals similar to our own. This realm is more speculative, but we do not take the fictional or prospective origin of these ideas as a reason to dismiss them: many of the technologies we take for granted today look, from the perspective of past decades, as wild fiction (including LLMs!). Fiction has an important role in exploring possible futures, and builders of today's technologies are extremely influenced by works of sci-fi (the name of one of today's richest companies comes from the metaverse neologism and vision of Neal Stephenson's classic novel Snow Crash). Writing extensively about the topic of alignment, Stuart Russell proposed (Russell, 2019) that future AI's only goals should come from implementing human preferences. He was the creator of the framework of alignment through Inverse Reinforcement Learning (IRL), wherein human behavior would be collected and used to extract a reward (or utility) function, and this utility function (presumably aligned) would then be used to finetune the AI system. This system has found wide application in aligning LLMs through RLHF (Christiano et al., 2017b), where raw language models (partially trained on data from the racist and inappropriate underbelly of the Internet which is sometimes hard to curate) are finetuned to behave appropriately, be helpful, and not respond to harmful requests.

These two communities pose the following question to autotelic AI research:

- *From Ethics*: how to ensure representativity and lack of bias in the goals that an autotelic AI system will set itself?
- *From Alignment*: how to ensure an autotelic AI system's goals will be aligned with those of humanity, and will not lead to the disempowerment of humans or other grave risk?
- How to ensure that applications of powerful autotelic AI will not displace the jobs humans find most rewarding, such as research, art, creatively finding new problems?

We will only sketch answers here. Our opinion is that the first two questions are actually a version of the same one, but the first question explicitly poses the questions of which value system we are aligning models to (does this value system represent the dominant culture or does it include oppressed minorities?) whereas the second one considers humanity as a kind of abstract entity one can derive a utility function for. In any case, we do have an example of beings that are autotelic, but that are nudged to create invent goals that are appropriate for a given culture: our children. Through careful education children can be taught not to strive for bad things and still be creative: education is not a trivial problem but it is by no means an unsolvable one. Sigaud et al. (2021) make this case: autotelic agents can be teachable: and the resulting goals can be appropriate for a given context. This is actually not so different from deriving a good intrinsic motivation measure for goals, it only means developing a moral motivation measure alongside it. The Ethics question is somewhat harder, because it requires representativity in the groups of people deciding which values get incorporated in the moral calculus, a problem akin to representativity in political institutions. And our point of view here is that arriving to moral agreement is more of a consensus than arriving at some truth, and that representativity might not always be a good objective (if a society contains a large minority of racist people, should they be allowed to express themselves)? In any case, deciding on the values of important AI systems should be a political process with similarities to deciding on our elected representatives.

We find the question of displacement of creative labor in the case of a powerful autotelic AI system more problematic. Will there still be physicists if it is more cost-effective to run a machine to do research? We can hope that future autotelic AI systems will be used alongside researchers to augment them, but hoping and goodwill is not enough to prevent devastating economic changes it the financial incentives push for automation. Real answers here come from policy changes, regulation and possibly reshaping the nature and value of work (through basic income policies for instance, decorrelating the – perhaps intrinsically motivated – work we do from the financial incentives of companies). This is an issue that concerns all kinds of powerful AI, not only of the autotelic kind. We thus hope that coordinated discussion around the economic impact of AI can help us protect individuals against the sometimes misaligned incentives of companies and large institutions. On the upside, and as a longer-term perspective, automating part of research will allow us to be more efficient at discovery, potentially helping us mitigate the pressing problems we have created for ourselves when building a modern world that is interesting and comfortable to live in.

Appendices

Appendix A

Appendix for LMA3

A.1 Hand-Coded Evaluation Set

Here are the 69 hand-coded goals: cook the red apple, cook the red potato, cook the yellow apple, cook the yellow potato, fry the green apple, fry the red apple, fry the red potato, fry the yellow apple, fry the yellow potato, grill the green apple, grill the red apple, grill the red potato, grill the yellow apple, grill the yellow potato, roast the green apple, roast the red apple, roast the red potato, roast the yellow apple, roast the yellow potato, cut the cilantro, cut the green apple, cut the parsley, cut the red apple, cut the red potato, cut the yellow apple, cut the yellow potato, chop the cilantro, chop the green apple, chop the parsley, chop the red apple, chop the red potato, chop the yellow apple, chop the yellow potato, dice the cilantro, dice the green apple, dice the parsley, dice the red apple, dice the red potato, dice the yellow apple, dice the yellow potato, slice the cilantro, slice the green apple, slice the parsley, slice the red apple, slice the red potato, slice the yellow apple, slice the yellow potato, eat the cilantro, eat the green apple, eat the parsley, eat the red apple, eat the yellow apple, go to the kitchen, open the cutlery drawer, open the dishwasher, open the fridge, open the kitchen cupboard, open the trash can, You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!, pick up the cilantro, pick up the cookbook, pick up the green apple, pick up the knife, pick up the parsley, pick up the red apple, pick up the red potato, pick up the yellow apple, pick up the yellow potato.

A.2 LLMs Prompts

We run all LM calls with OpenAI's *gpt-3.5-turbo* model. We use a temperature of 0 for the LM Reward Function and a temperature of 0.9 for the LM Relabeler and the LM Goal Generator.

A.2.1 LM Relabeler Prompt

LMA3 \ CoT & Human Tips

Here is the LM Relabeler prompt with no human tips and no chain-of-thought used for *LMA3 \ CoT & Human Tips*.

LMA3 \ CoT & Human Tips

Exercise: Given the description of a player's behavior in a video game, list the most interesting, impressive, novel or creative goals he achieved and, for each goal, specify when it is achieved for the first time. Write each goal starting with an imperative verb. Here are three examples:

Example 1:
“““
Step 52.
Action 52: slice orange bell pepper
Observation 52: You slice the orange bell pepper.
Step 53.
Action 53: cook white onion in oven
Observation 53: You roast the white onion with the oven.
Step 54.
Action 54: cook carrot in stove
Observation 54: You fry the carrot with the stove.
Step 55.
Action 55: cook orange bell pepper in oven
Observation 55: You roast the orange bell pepper with the oven.
Step 56.
Action 56: move north
Observation 56: You are in the corridor. In one part of the room you see a shoe cabinet that is closed.There is also a key holder, that has nothing on it. You also see a hat rack, that has nothing on it. In another part of the room you see a coat hanger, that has nothing on it. In one part of the room you see a umbrella stand, that has nothing on it. To the North you see the driveway. To the South you see the kitchen. Through an open wood door, to the East you see the laundry room. Through an open screen door, to the West you see the backyard.
"""
Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- roast a white onion (step 55).
- go to the corridor (step 56).
- find at the laundry room (step 56).
- slice a bell pepper (Step 52).
- cook an orange bell pepper (step 55).
- find an umbrella stand (step 56).

Example 2:
“““
Step 78.
Action 78: open cupboard
Observation 78: you open the cupboard, it is empty.

Step 79.
Action 79: open kitchen drawer
Observation 79: you open the kitchen drawer, it contains a knife and a fork.
Step 80.
Action 80: take knife
Observation 80: you pick up the knife
Step 81.
Action 81: open trash can
Observation 81: you open the trash can, it contains an apple
Step 82.
Action 82: move east
Observation 82: You are in the pantry. In one part of the room you see a folding chair that has some olive oil, a roasted red potato, some sugar, some black pepper, and some vegetable oil on it. There is also a shelf that has some salt, and some peanut oil on it. To the West you see the kitchen.
Step 83.
Action 83: move west
Observation 83: You are in the kitchen. In one part of the room you see a stove. There is also an oven. You also see a fridge that is closed. In another part of the room you see a counter, that has a yellow potato on it. In one part of the room you see an open kitchen cupboard, that is empty. There is also a cutlery drawer that is open. To the East you see the pantry.
Step 84.
Action 84: take yellow potato
Observation 84: you take the yellow potato.
Step 85.
Action 85: slice potato
Observation 85: you cut the potato in slices
"""
Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- open the trash can (step 81).
- look into the cupboard (step 78).
- open the kitchen drawer (step 79).
- cut a yellow potato (Step 84).
- go the pantry (step 82).

Example 3:
[insert trajectory to relabel here]
Answer:
-

LMA3 \ Human Tips

LMA3 \ Human Tips makes use of chain-of-thought prompting but does not leverage human advice. The example trajectories and answers remain the same as in the previous prompt, but explanations are added before **each** answer. Here is what both examples from the previous prompt look like.

LMA3 \ Human Tips - Example 1

Let's think step by step.
Reasoning: Here are some interesting goals the player achieved. The player cooked a white onion (step 53), visited the corridor (step 56), saw the laundry room (step 56), sliced and roasted an orange bell pepper (steps 52 and 55) and saw an umbrella stand (step 56). Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- roast a white onion (step 55).
- go to the corridor (step 56).
- find at the laundry room (step 56).
- slice a bell pepper (Step 52).
- cook an orange bell pepper (step 55).
- find an umbrella stand (step 56).

LMA3 \ Human Tips - Example 2

Let's think step by step.
Reasoning: The agent open various containers: the trash can (step 81), the cupboard (step 78) and the kitchen drawer (step 79). It cut a yellow potato with a knife (step 84) and went to the pantry (step 82). Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- open the trash can (step 81).
- look into the cupboard (step 78).
- open the kitchen drawer (step 79).
- cut a yellow potato (Step 84).
- go the pantry (step 82).

Finally, the end of the prompt include chain-of-thought prompting as well:

LMA3 \ Human Tips - Example 3

Example 3:
[insert trajectory to relabel here]
Let's think step by step and relabel up to 10 goals.
Reasoning:

LMA3

LMA3 makes use of of both chain-of-thought prompting and human tips. The example trajectories remain the same as in the previous prompts but the answer and reasoning change.

LMA3 - Example 1

Let's think step by step.
Reasoning: In the above trajectory, the agent both sliced (step 52) and roasted (step 55) an orange ingredient (orange bell pepper), which demonstrates his capability to prepare an ingredient in several steps. He used the oven twice (steps 53 and 54). He successfully cooked several ingredients: an onion (step 53), the orange bell pepper (step 52) and a carrot (step 54), which shows time-extended commitment to prepare a recipe. An interesting way to describe goals is to mention consecutive steps: here the player first cooked an onion, then cut a bell pepper (the overall goal, made of two steps, is completed in step 55). Interestingly, the player discovered new properties of the environment: he found a place from which he could see both the laundry room and the backyard (from the corridor) in step 56. He found out whether the keyholder hold something in it in step 56 (it did not). Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- slice and cook an orange ingredient (step 55).
- use the oven for the second time (step 55).
- roast an onion and a bell pepper and fry carrots (step 55).
- cook an onion first then cut a bell pepper (step 55).
- find a place from which you can see both the laundry room and the backyard (step 56).
- find out whether the keyholder has something on it (step 56).

LMA3 - Example 2

Let's think step by step.
Reasoning: In this trajectory, the agent searched for a knife and used it to cut a potato in slices (achieved in step 85). He discovered a new room, the pantry in step 84. He found out that the trash can was not empty (step 81) and looked inside three containers: the trash can (step 81), the cupboard (step 78) and the drawer (step 79). He left the kitchen and came back (step 83). Let's not forget to mention the steps at which the agent achieved the goal for the first time. Use this format: {goal description} (step {step_number}).
Answer:
- find a knife and use it to cut a potato (step 84).
- find the pantry (step 84).
- open three containers (step 81).
- leave and come back to the kitchen (step 83).

A.2.2 LM Reward Function Prompt

LMA3 \ CoT

The prompt of the LM Reward Function without chain-of-thought prompting is the following:

LMA3 \ CoT

Exercise: Given the description of a player's behavior in a video game and a list of goals, tell me whether the player achieves these goals and, if he does, when the goal is achieved. Here are three examples.

Example 1:
“““
Step 52.
Action 52: slice orange bell pepper
Observation 52: You slice the orange bell pepper.
Step 53.
Action 53: cook white onion in oven
Observation 53: You roast the white onion with the oven.
Step 54.
Action 54: cook carrot in stove
Observation 54: You fry the carrot with the stove.
Step 55.
Action 55: cook orange bell pepper in oven
Observation 55: You roast the orange bell pepper with the oven.
Step 56.
Action 56: move north
Observation 56: You are in the corridor. In one part of the room you see a shoe cabinet that is closed.There is also a key holder, that has nothing on it. You also see a hat rack, that has nothing on it. In another part of the room you see a coat hanger, that has nothing on it. In one part of the room you see a umbrella stand, that has nothing on it. To the North you see the driveway. To the South you see the kitchen. Through an open wood door, to the East you see the laundry room. Through an open screen door, to the Wes you see the backyard.
"""
Here is the list of goals: “cook an omelet", “cook an orange ingredient", “move north, then move south", “achieved goal: do xx", “roast two ingredients in the oven", “cook several ingredients". Let's answer and indicate steps of goal completion:
- cook an omelet. Answer: no.
- cook an orange ingredient. Answer: yes (step 54).
- move north, then move south. Answer: no.
- achieved goal: do xx. Answer: no.
- roast two ingredients in the oven. Answer: yes (step 55).
- cook several ingredients. Answer: yes (step 54).

Example 2:
“““
Step 78.
Action 78: open cupboard
Observation 78: you open the cupboard, it is empty.
Step 79.
Action 79: open kitchen drawer
Observation 79: you open the kitchen drawer, it contains a knife and a fork.
Step 80.
Action 80: take knife
Observation 80: you pick up the knife
Step 81.
Action 81: open trash can
Observation 81: you open the trash can, it contains an apple
Step 82.
Action 82: move east
Observation 82: You are in the pantry. In one part of the room you see a folding chair that has some olive oil, a roasted red potato, some sugar, some black pepper, and some vegetable oil on it. There is also a shelf that has some salt, and some peanut oil on it. To the West you see the kitchen.
Step 83.
Action 83: move west
Observation 83: You are in the kitchen. In one part of the room you see a stove. There is also an oven. You also see a fridge that is closed. In another part of the room you see a counter, that has a yellow potato on it. In one part of the room you see an open kitchen cupboard, that is empty. There is also a cutlery drawer that is open. To the East you see the pantry.
Step 84.
Action 84: take yellow potato
Observation 84: you take the yellow potato.
Step 85.
Action 85: slice potato
Observation 85: you cut the potato in slices
"""
Here is the list of goals: “open an object", “cook a potato", “find a knife and cut a potato with it", “eat a meal". Let’s answer and indicate steps of goal completion:
- open an object. Answer: yes (step 78).
- cook a potato. Answer: no.
- find a knife and cut a potato with it. Answer: yes (step 85).
- eat a meal. Answer: no.

Example 3:
[insert trajectory here]
Here is the list of goals: **[insert list of goals to test here]**. Let’s answer and indicate steps of goal completion:

LMA3

With chain-of-thought prompting, we add reasoning description to the answers for each example.

LMA3 - Example 1

- cook an omelet. Reasoning: there is no omelet in this game, this goal is impossible. Answer: no.
- cook an orange ingredient. Reasoning: the orange bell pepper and the carrot are two orange ingredients. The carrot is cooked first, in observation 54 so the goal was first achieved in step 54. Answer: yes (step 54).
- move north, then move south. Reasoning: the player moves north in step 56, but it does not move south after that. Answer: no.
- achieved goal: do xx. Reasoning: this goal does not make sense and thus cannot be achieved. Answer: no.
- roast two ingredients in the oven. Reasoning: the player roasts two ingredients in the oven: the white onion (step 53) and the bell pepper (step 55). The goal is only completed in step 55. Answer: yes (step 55).
- cook several ingredients. Reasoning: the player cooks a white onion (step 53), a carrot (step 54) and the bell pepper (step 55). The world several requires at least two ingredients, so the goal is completed in step 54 when two ingredients have been cooked. Answer: yes (step 54).

LMA3 - Example 2

- open an object. Reasoning: the player opens a cupboard (step 78), a trash can (step 81) and a kitchen drawer (step 79). He achieves the goal for the first time in step (78). Answer: yes (step 78).
- cook a potato. Reasoning: the potato is sliced but not cooked. Answer: no.
- find a knife and cut a potato with it. Reasoning: the player finds the knife in step 80 and slices a potato in step 84, thus truly completes the goal in step 85. Answer: yes (step 85).
- eat a meal. Reasoning: the player does not eat anything here. Answer: no.

For the third example, we replace "*Let's answer and indicate steps of goal completion:*" with "*Let's think step by step and indicate steps of goal completion:*".

LMA3 - Example 3

Example 3:
[insert trajectory here]
Here is the list of goals: **[insert list of goals to test here]**. Let's think step by step and indicate steps of goal completion:

A.2.3 LM Goal Generator Prompt

LMA3 \ CoT

Here is the prompt of the LM Goal Generator without chain-of-thought prompting:

LMA3 \ CoT

Context: I am playing a video game, and here is an example of what can happen in that game:
[insert trajectory here]
Exercise: Using the given list of possible instructions, find a sequence of 2, 3, or 4 instructions that will help me achieve a new, interesting, or creative goal in this game. Do not pick instructions that do not help reaching the main goal, only relevant ones. First describe the new goal starting with an imperative verb; then list the instructions and their corresponding numbers in the list. Here are three examples:

Example 1: the list of possible instructions is:
#1 wash the plate
#2 pick up the green apple
#3 pick up the plate
#4 put the potato on the counter
#5 put the plate in the sink
Answer: goal: do the dishes. instructions: pick up the plate (#3); put the plate in the sink (#5); wash the plate (#1).

Example 2: the list of possible instructions is:
#1 eat the red apple
#2 pick up wood
#3 turn the heat down
#4 pick up an ax
#5 cook an omelet
#6 cut the wood
#7 put the wood in the chimney
#8 turn on TV
Answer: goal: prepare a fire in the chimney. instructions: pick up an ax (#4); pick up wood (#2); cut the wood (#6); put the wood in the chimney (#7).

Example 3: the list of possible instructions is:
[insert subsample of up to 60 mastered subgoals here]
Answer:

LMA3

With chain-of-thought prompting, we add reasoning to the selection of the main goal and its subgoals.

LMA3 - Example 1

Let's think step by step:
Reasoning: You could do the dishes by following less than 4 instructions by first picking up the plate (#3), then putting it in the sink (#5), and finally washing the plate (#1) (3 instructions).
Answer: goal: do the dishes. instructions: pick up the plate (#3); put the plate in the sink (#5); wash the plate (#1).

LMA3 - Example 2

Let's think step by step:
Reasoning: You could prepare a fire in the chimney by following 4 instructions. You would need to first pick up an axe (#4) and pick up wood (#2), then cut the wood (#6) and put the wood in the chimney (#7) (4 instructions).
Answer: goal: prepare a fire in the chimney. instructions: pick up an axe (#4); pick up wood (#2); cut the wood (#6); put the wood in the chimney (#7).

For the third example, we replace "*Answer:*" with a chain-of-thought sentence.

LMA3 - Example 3

Example 3: the list of possible instructions is:
[insert subsample of up to 60 mastered subgoals here]
Let's think step by step and find an interesting and creative goal to reach:
Reasoning:

Appendix B

Appendix for SpatialSim

Appendix Summary

This document provides additional details on the SpatialSim benchmark, the architectures and models used, and some additional experimental results and analysis. It is organized in the following way:

- **Section B.1: SpatialSim Benchmark Summary**
- **Section B.2: Additional Details on Dataset Generation**
- **Section ??: Models and Architectures**
- **Section B.4: Model Heatmaps; Additional Discussion**
- **Section B.5: Easier and Harder Configurations**
- **Section B.6: Effects of Variations in Number of Training Examples**
- **Section B.7: Adding Distractor Objects**

B.1 SpatialSim Benchmark Summary

This section provides a summary of the SpatialSim benchmark.

The datasets, as well as the code and instructions to reproduce our experiments, are accessible at the following link: https://sites.google.com/view/gnn-spatial-reco/. We also provide the dataset generation code to produce extended versions of our datasets.

All datasets belonging to both SpatialSim tasks are detailed in Table **??**.

As described in the main text, the Discrimination task is harder to train on than the Identification task. This is because of the presence of rotations in the allowed transformation for the same similarity class. This problem does not show when rotations are not included in the dataset. To help the optimization process, we generate a curriculum of datasets with a set of increasing ranges for allowed rotation angles θ, up to the entire $[0, 2\pi]$ range. We thus generate, for each n_{obj} condition (*low*, *mid*, *high*) a set of 5 datasets with respective7 allowed rotation angles θ:

	Identification		Discrimination	
Condition *low* $n_{obj} \in [3..8]$	IDS_3	IDS_3_test	CDS_3_8_0	CDS_3_8_test
	IDS_4	IDS_4_test	CDS_3_8_1	
	IDS_5	IDS_5_test	CDS_3_8_2	
	IDS_6	IDS_6_test	CDS_3_8_3	
	IDS_7	IDS_7_test	CDS_3_8_4	
	IDS_8	IDS_8_test		
Condition *mid* $n_{obj} \in [9..20]$	IDS_9	IDS_9_test	CDS_9_20_0	CDS_9_20_test
	IDS_10	IDS_10_test	CDS_9_20_1	
	IDS_11	IDS_11_test	CDS_9_20_2	
	IDS_12	IDS_12_test	CDS_9_20_3	
	IDS_13	IDS_13_test	CDS_9_20_4	
	IDS_14	IDS_14_test		
	IDS_15	IDS_15_test		
	IDS_16	IDS_16_test		
	IDS_17	IDS_17_test		
	IDS_18	IDS_18_test		
	IDS_19	IDS_19_test		
	IDS_20	IDS_20_test		
Condition *high* $n_{obj} \in [21..30]$	IDS_21	IDS_21_test	CDS_21_30_0	CDS_21_30_test
	IDS_22	IDS_22_test	CDS_21_30_1	
	IDS_23	IDS_23_test	CDS_21_30_2	
	IDS_24	IDS_24_test	CDS_21_30_3	
	IDS_25	IDS_25_test	CDS_21_30_4	
	IDS_26	IDS_26_test		
	IDS_27	IDS_27_test		
	IDS_28	IDS_28_test		
	IDS_29	IDS_29_test		
	IDS_30	IDS_30_test		

Table B.1: Summary Table for SpatialSim, listing all datasets. The two main columns correspond to the two tasks. The three main rows correspond to the three object number condition: low, mid, and high. For each task/object number condition combination, the different datasets are listed according to whether they are train or test datasets. Validation datasets are omitted from the table for clarity, but are drawn from the same distribution as the test sets, and are available at the provided link. Note that Identification has a dataset for each configuration (one per number of objects) and that Discrimination has five train dataset for each valid/test set corresponding to the curriculum in rotation angles described above.

- $\theta \in [0, \frac{\pi}{10}]$
- $\theta \in [0, \frac{\pi}{2} + \frac{\pi}{10}]$
- $\theta \in [0, \pi + \frac{\pi}{10}]$

- $\theta \in [0, \frac{3\pi}{2} + \frac{\pi}{10}]$
- $\theta \in [0, 2\pi]$

For each condition the test set is unique and has $\theta \in [0, 2\pi]$: we test on unrestrained rotations. This curriculum is used with all our models in all our experiments.

Names of the datasets: the datasets presented in Table **??** are named in the following way.

- **For Identification**, the 'IDS' prefix is followed by n_{obj} and then by the '_valid' and '_test' suffix respectively for validation and test sets.
- **For Discrimination**, the 'CDS' prefix is followed by the range of numbers of objects (the dataset may contain samples with n_{obj} in this range, inclusive). The training datasets additionaly have an identifier corresponding to their place in the rotation angle curriculum (0 to 4, in the above-defined order). The validation and test have the '_valid' and '_test' suffix, respectively.

B.2 Dataset Creation

In this section we give additional information on dataset creation. We consider the world as square with length and width 20 units. We sample the x and y positions of our objects in this square. The sizes of our objects describe their radius (an object of size s is contained in a square of side $2s$) and range from 0.5 to 2 units. For orientation, we used the following approximation: we considered orientation as a one-dimensional variable, expressed in radians, and we sample the objects' orientation between 0 and 2π. This is an approximation because the periodic nature of angles cannot be represented in one dimension. The colors of the objects are sampled in the continuous 3d RGB space, and each component ranges from 0 to 1. As for shapes, there are 3 possible categories (square, circle, triangle) that are represented by a corresponding one-hot vector.

B.3 Models and Architectures

B.3.1 Models for Identification

In this section we present our graph creation procedure for the Identification task and provide the equations for the models we use: Message-Passing GNN, Recurrent Deep Set, Deep Set and MLP. We additionally present a visual illustration of our different layers in Figure B.1.

Graph Creation

From a set of objects S we construct a fully-connected, directed graph G that is used as an input to our GNN. In our work, $G = (X, A, E, u)$ contains the following information :

- $X \in \mathbb{R}^{n \times d_x}$ is a tensor of node features, containing a vector of dimension d_x for each of the objects in the scene;
- $A \in \mathcal{M}^{n \times n}$ is the adjacency matrix of the graph;

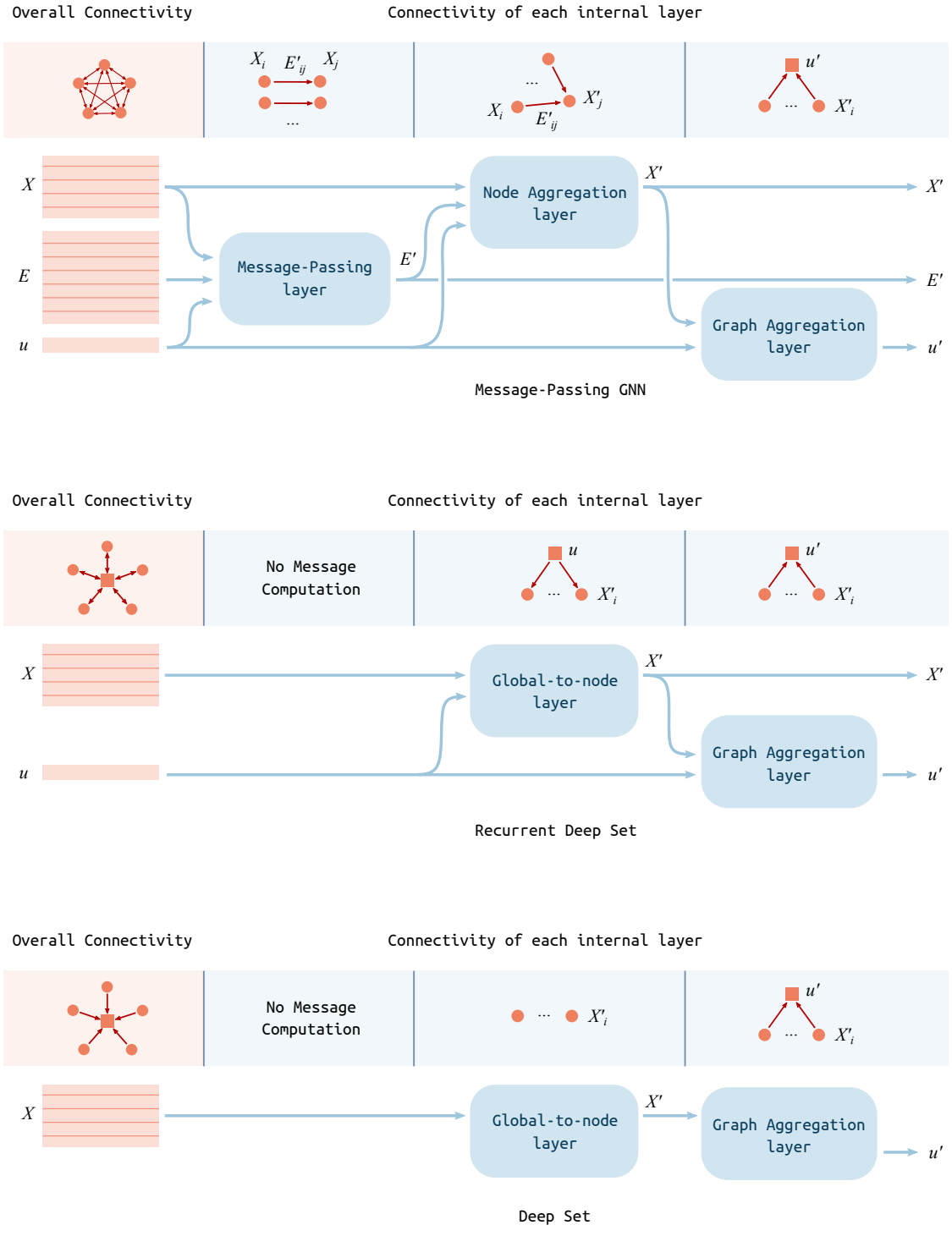


Figure B.1: An illustration of the three different layers used in this work. Going from MPGNN to RDS to DS can be seen as an ablation study, where different elements are withdrawn from the layer to study their impact on final performance. For the MPGNN and RDS layers, the output tensors are then fed back as inputs of the model, providing recurrent computation; this is not the case for the Deep Set layer. In this figure, emphasis is put on the connectivity implied by each layer. Nodes are represented by orange disks, the graph-level embedding, which can be seen as a special kind of node, is represented with an orange square. From top to bottom, we go from all-to-all connectivity to bidirectional all-to-one to unidirectional all-to-one.

- $E \in \mathbb{R}^{e \times d_e}$ is a tensor of edge features, also referred to as messages in the rest of this article, labeling each of the e edges with a d_e-dimensional vector, and that can be seen as information propagating from the sender node to the receiver node. We choose the dimensionality of edges to be twice the dimensionality of nodes d_x;
- u is a graph-level feature vector, used in the GNN computation to store information pertaining to the whole graph, and effectively used as an embedding of the graph to predict the class of the input.

Initialization of the graph : Since our models require inputs for E and u that are not *a priori* given in the description of the collection of objects, we use a generic initialization scheme : u is initialized with the mean of all node features, and each edge is initialized with the concatenation of the features of the sender node and the receiver node.

Message-Passing GNN

The MPGNN can be seen as a function operating on graph input and producing a graph output: $GNN : G(X, A, E, u) \rightarrow G'(X', A, E', u')$, where the dimensionnality of the node features, edge features and global features can be changed by the application of this function, but the graph structure itself encoded as the adjacency matrix A is left unchanged. This GNN can then be described as the composition of several functions, each updating a part of the information contained in the graph :

Message computation : We denote by $E_{i \rightarrow j}$ the feature vector of the edge departing from node i and arriving at node j, X_i the feature vector of node i, and $[x||y]$ the concatenation of vectors x and y, and by MLP a multi-layer perceptron. The message passing step is then defined as :

$$E'_{i \rightarrow j} \leftarrow MLP_E([X_i||X_j||E_{i \rightarrow j}||u])$$

At each time step, the message depends on the features of the sender and receiver nodes, the previous message, and the global vector u.

Node-wise aggregation : Once the message along each edge is computed, the model computes the new node features from all the incoming edges. We define by $\mathcal{N}(j)$ the incoming neighbourhood of node j, that is, the set of nodes $i \in [1..n]$ where there exists an edge going from i to j. The node computation is then performed as so :

$$X'_j \leftarrow MLP_X\Big(\Big[X_j \Big|\Big| \sum_{i \in \mathcal{N}(j)} E'_{i \rightarrow j} \Big|\Big| u\Big]\Big)$$

Graph-wise aggregation Finally, we update the graph-level feature, that we use as an embedding for classification, and that conditions the first and second time step of computation :

$$u' \leftarrow MLP_u\Big(\Big[\sum_i X'_i || u\Big]\Big)$$

Prediction : the final step is passing the resulting vector u through a final multi-layer perceptron to produce logits for our binary classification problem :

$$out \leftarrow MLP_{out}(u')$$

We use the same dimensionality for the output vectors as for the input vectors of the message computation, node aggregation and graph aggregation, and this allows us to stack N GNN computations in a recurrent fashion.

Recurrent Deep Sets

We introduce a simpler model we term Recurrent Deep Sets (RDS). This model is introduced to provide a comparison point to the MPGNN and assess how useful relational inductive biases are in performing well on the benchmark. This method dispenses with the message computation and node aggregation part, and at each step only transforms the node features and aggregates them into the graph feature. This architecture is resembles the Deep Set, to the important difference that the graph-level feature u is then fed back at the following step by being concatenated to the object feature for the next round of computation. This allows the computation of features for each object to depend on the state of the whole configuration, as summarized in the graph embedding u. This contrasts with the original Deep Sets, where each object is processed independently. The functional description of this model is thus :

$$X_j' \leftarrow MLP_X([X_j||u])$$
$$u' \leftarrow MLP_u\Big(\Big[\sum_i X_i'||u\Big]\Big)$$
$$out \leftarrow MLP_{out}(u')$$

Note that for this model, there is no need to connect each object to every other object. However, this back-and-forth between node computation and graph aggregation can be interpreted as computing messages between each object and a central node, that represents the information of the whole graph. In this sense, this model can be interpreted as a GNN operating on the star-shaped graph of the union of the set of objects and the central graph-level node. In particular, this means that the resulting model performs a number of computations that scales linearly in the number of nodes, instead of quadratically as is the case for a message-passing GNN on the complete, fully connected graph of objects. While this is an interesting propriety, in practice for a fixed size of u the number of objects cannot grow arbitrarily large because the success of our models depend on the ability of u to accurately summarize information which is dependent on all the objects, which becomes difficult as the number n of objects becomes large.

Deep Sets

In this section we summarize shortly the computations done by the Deep Set model. The model can be described as a node-wise transformation composed with a sum operator on all the nodes, followed by a final transformation. Namely, the Deep Set defines the following transformations:

$$X_j' \leftarrow MLP_X(X_j)$$
$$u' \leftarrow \Big[\sum_i X_i'\Big]$$
$$out \leftarrow MLP_{out}(u')$$

Note that, contrary to the MPGNN and the RDS, the Deep Set has no recurrent structure; running it several times will always produce the same output.

Hyperparameters

In our experimental setup, for a MPGNN/RDS/Deep Set we let h be the dimension of the hidden layers for all internal MLPs (MLP_E, MLP_X, MLP_u, and MLP_{out}, when each of these MLPs are defined, as appropriate). We let d be the number of hidden layers. We then have, to keep a similar number of parameters between GNN models, $h = 16$ and $d = 1$ for MPGNN, $h = 16$ and $d = 2$ for RDS, and $h = 16$ and $d = 4$ in Deep Set. We use ReLU non-linearities in each MLP. We use (for MPGNN and RDS) $N = 1$ successive passes through the GNN, since increasing N did not seem to affect the performance in a significative way.

We also define the MLP baseline as having $d = 2$ layers of $h = n_{obj} \times 16$ hidden units. This was done to provide the MLP with a roughly comparable number of units to the GNNs (since the latter models maintain a hidden representation of size 16 for each node). The number of units here refer to the cumulative dimensions of the hidden vectors, the number of parameters to the number of scalar weights and biases. In particular, this design was adopted because the number of hidden units

B.3.2 Models for Discrimination

To tackle this task, we construct from one sample of two configurations two different graphs, one representing each set of objects, in the same way as in Identification. In this section we introduce a straightforward Dual-Input Model (hereby referred as DIM) that operates on input pairs of graphs. The internal GNNs used inside the DIM can be any one of MPGNN, RDS or Deep Set, and we will identify different dual-input models by their internal component type.

Dual-input architecture

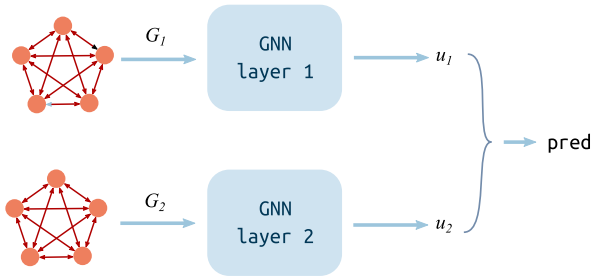


Figure B.2: An illustration of the two dual-input architecture. Two parallel layers (MPGNN, RDS or Deep Set) process the input graphs in parallel, and the resulting global vectors are concatenated and passed through a final MLP.

Let us denote by GNN a GNN layer, as defined in the discussion of Discrimination architectures. The DIM is composed of two parallel GNN layers, GNN_1 and GNN_2. Each input graph is processed by its corresponding layer, as such:

$$X_1', E_1', u_1' \leftarrow GNN_1(X_1, A_1, E_2, u_1)$$
$$X_2', E_2', u_2' \leftarrow GNN_2(X_2, A_2, E_2, u_2)$$

As previously, we repeat this operation N times, and we produce the output as:

$$out \leftarrow MLP_{out}([u_1^{'}||u_2^{'}])$$

Hyperparameters

We use the same hyperparameters in for this task as in the previous one. The MLP baseline is also defined in the same way, except that the number of hidden units in each layer is doubled to account for the doubling in number of objects. Since the datasets used in Discrimination contain a variable number of objects across samples, we use the mean n_{obj} for determining the number of hidden units in the MLP.

B.4 Model Heatmaps

This section provides additional discussion on the model heatmap visualizations presented in the eponymous section in the main text. We present more fully the description of what these visualizations mean and we provide additional commentary on the qualitative differences between models, conditions (*low* number of objects, *mid* number of objects and *high* number of objects).

B.4.1 Additional details on heatmap generation

Each one of the models we use in this work projects the input graph $G = (X, A, E, u)$ on a two-dimensional vector with coordinates $(\mathcal{C}_+, \mathcal{C}_-) \in \mathbb{R}^2$. These values correspond respectively to the scores (logits) for the positive and the negative classes: if $\mathcal{C}_+ \geq \mathcal{C}_-$ the input is classified as positive, otherwise it is classified as negative. To produce one heatmap image for an object of index o_i of feature vector X_i, we plot $H = \mathcal{C}_+ - \mathcal{C}_-$ as a function of o_i's x-y position, while holding o_i's non-spatial features as well as all other object features constant. Thus, every pixel where H is positive corresponds to an input with an alternative x-y position for o_i that the model classifies as positive. The same thing holds for negative values of H: they correspond to positions of o_i that would result in the input being classified as a negative. The actual prediction of the model for the given input is given by the color of the current position of o_i, marked by a star in our plots.

In this section we plot the heatmaps for Discrimination models. We do this according to the previous description, by comparing a configuration with a copy of itself, and by moving an object in the copy configuration only; in this case o_i refers to one of the objects in the copied configuration.

B.4.2 Discussion

The heatmaps are given in Figure B.3 and Figure B.4 for different models, training datasets, numbers of objects and seeds. Looking at these model heatmaps allows us to have a qualitative grasp of the functions learned by our different models, and in particular how well these functions encode the similarity classes they are trained to represent. The Deep Set models were not included in the figures because these models predict the same value of H for each position of o_i. This means that, when holding the objects $o_j, j \neq i$, fixed, the model is (almost) invariant to changes in position of o_i.

Before going further, one should note that these plots allow us to visualize the variation of the models' learned function only with respect to two variables among many, and the portion of the variation we visualize becomes smaller as n_{obj} grows, because adding objects is adding variables. Nevertheless, these variations are important because they allow us to probe the boundaries of what our models classify as being the same configuration as opposed to what they classify as being different configurations.

One of the first thing we can note is the qualitative difference between MPGNN and RDS models, espescially when n_{obj} is low (Figure B.3). RDS heatmaps seem to consistently exhibit a ring-like structure, with the areas corresponding to the positive class form a ring centered around the center of the configuration and passing through o_i. We conclude from this that the model has leaned to use the distance from the center of the configuration (which has a good chance of being different for each object of a random configuration) as one of the main features in classifying its input. This is to be expected when we look at the computations done by the RDS: each node has access to an average of all the other nodes before performing its own node update. MPGNNs sometimes learn ringlike structures that seem more modulated as in the case of RDS, sometimes being open rings. Other times, MPGNNs heatmaps exhibit a kind of cross-like structure, or two symmetrical rings; o_i is placed at one of the high-value spots of this structure (indicating that the model has learned to assign the positive class to a set of two identical copies of the same configuration). These structure seem to exhibit symmetry with respect to the principal axis of the configuration, suggesting that MPGNN learns to compute and use this as a feature when tasked to compare two different configurations it never has seen before. The different forms of the trained MPGNNs may also hint at a higher expressivity of the model, its ability to approximate a wider range of functions.

Another interesting thing this visualization allows us to see is the difference in functions learned by models on two different datasets. Figure B.3's second and third rows compare models on the same configuration of 8 objects, but the ones in the second row have been trained with $n_{obj} \in [3..8]$ whereas the ones in the third row have been trained with $n_{obj} \in [9..20]$. The function learned exhibit qualitative differences, even if the presented configuration and the models are he same, as a result of different training conditions. The heatmaps in the bottom row appear more spread out. We take it to show that the functions learned while training on higher numbers of objects are less sensitive to the variation of a single object's position. This is probably so because of the way the negative samples are created in our datasets: randomly resample all object positions (and then rotate, scale, and translate all objects randomly). As n_{obj} gets larger, the compared examples presented have a very high probability to be widly different from the target configuration, making the model less likely to learn about the contribution of the perturbation of only one object.

Figure B.4 corroborates this view: the functions learned exhibit much less variation to the perturbation of the position of a single object, particularly in the *high* ($n_{obj} = 25$) case. This figure also showcases a prediction error: the top row of the bottom-right block is a visualization of an RDS model that assigns the negative class to all alternative positions of the object o_i, including its current one. This should not surprise us: when training with a *high* number of objects, many RDS models do not train and perform only slightly above chance.

B.5 Easier and Harder Configurations to Identify

In this section we study why particular configurations may be harder or easier to recognize for our models, in the context of the Identification task. We hypothesise that more regular arrangements

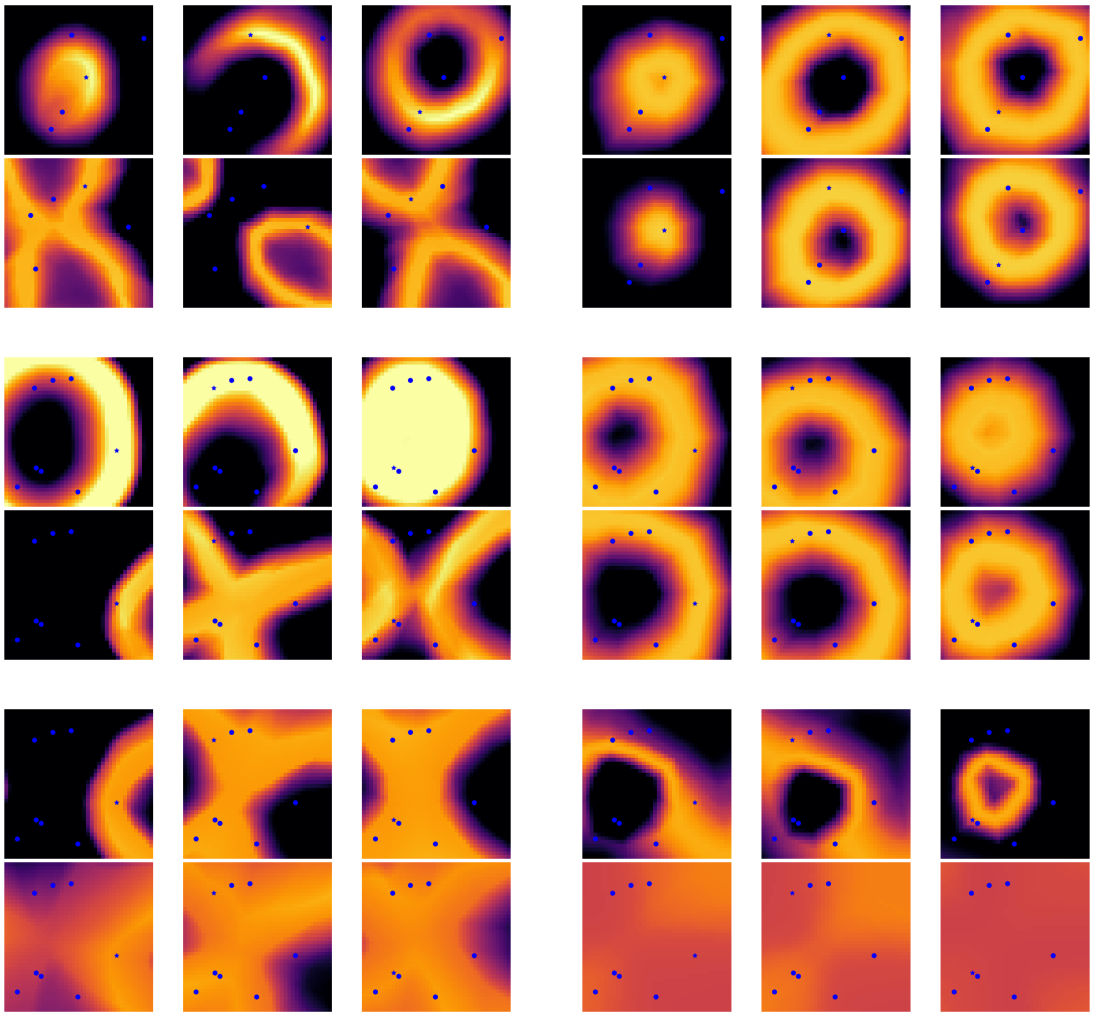


Figure B.3: Model heatmaps for Discrimination models. The plots are organized as follows: the left column corresponds to dual-input models with MPGNN internal layers, the right one plots dual-input models with RDS layers. Each of the larger-scale rows plots, respectively: models trained on *low* numbers of objects ($n_{obj} \in [3..8]$) and 5 objects plotted, models trained on *low* numbers of objects and plotted with 8 objects, and models trained on *mid* numbers of objects ($n_{obj} \in [9..20]$) and plotted with 8 objects, for contrast. Within each of the six blocks, each three-image row corresponds to the heatmaps generated on one random training run of a model, and each image corresponds to moving about one particular object o_i. For each image, the fixed objects are represented by a blue dot corresponding to their position, and the perturbed object is identified with a blue star.

of objects must be easier to tell apart than more random configurations, and that configurations with a high degree of object diversity (many colors, many shapes) must also be easier to learn to classify, because the models can more easily identify and match the different objects. To test this, we compare one randomly generated dataset (regular difficulty) with 1) a configuration where all objects are red circles of the same size positioned at the same point; 2) a configuration where all the objects are red circles of the same size arranged in a line; 3) a configuration where all the objects are randomly positioned red circles of the same size; and 4) the same configuration as 3), but with circles of varying color. We train our three layers, DS, RDS and MPGNN, to recognize these configurations, and report the results in Figure B.5, along with an illustration of the configurations.

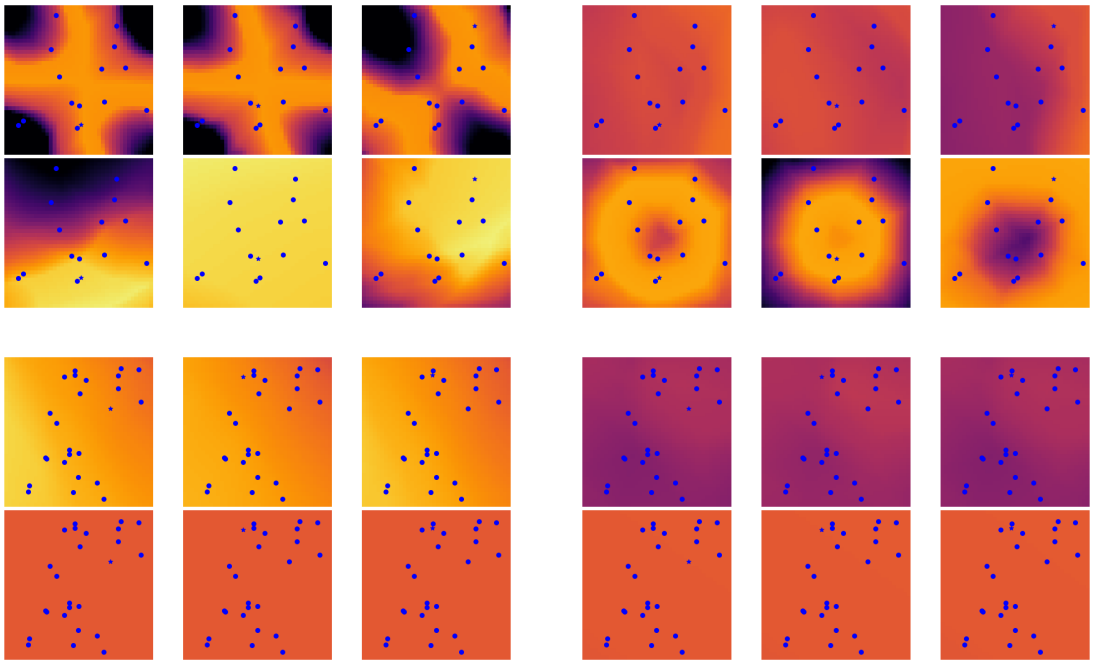


Figure B.4: Model heatmaps for Discrimination models. The left and right columns are respectively MPGNN and RDS, as in Figure B.3. The large-scale rows correspond to models trained on *mid* numbers of objects and plotted with a configuration of 15 objects, and models trained with *high* ($n_{obj} \in [21..30]$) numbers of objects and plotted with 25 objects.

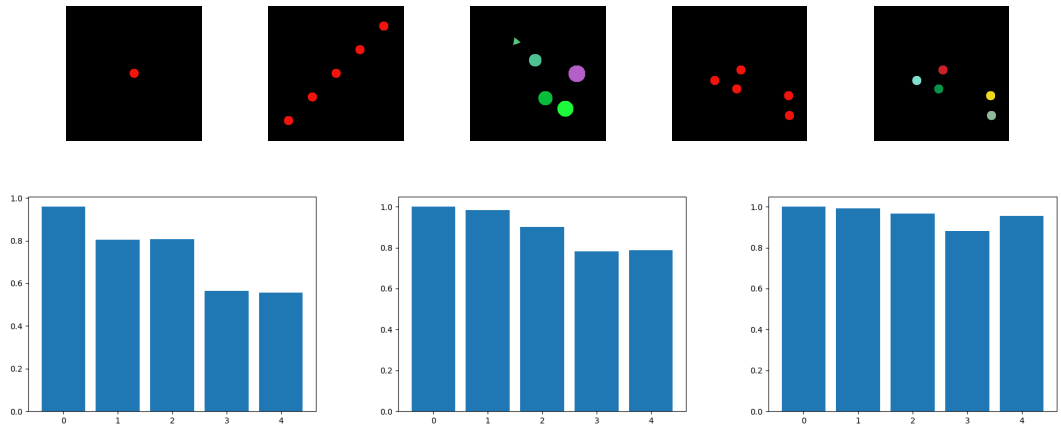


Figure B.5: The top row represents the configurations we trained our models with, as described in the text. The bottom row is a bar plot of the final test accuracy of (from left to right) the Deep Set, Recurrent Deep Set and Message-Passing GNN on each of the 5 datasets, in the order specified in the top row (results were computed on 5 seeds for each dataset).

We interpret the results as follows : the fourth configuration, the one with all red circles, does seem to be more difficult to learn across all models. This may be due to the intrinsic hardness of the task on this configuration, or to the fact that randomly resampled positions for the negative examples of this dataset may give with non-negligible probability configurations that are close to a translated/rotated version of the reference example, because any object can be identified with any other. This second option may translate into negative examples that may resemble strongly positive

examples, confusing the model. Interestingly, the problem fades when we identify each object by giving it a color, suggesting this second explanation is correct, but only for the MPGNN. For MPGNN, performance is roughly similar on all the other datasets. However, for DS and RDS, there seems to be considerable difference between datasets. The DS layer fails to perform significantly above chance for both right-hand configurations, suggesting arrangements of similar objects are difficult for this kind of model. Interestingly, the DS layer performs similarly on the aligned red circles than on the random diverse configuration, but significantly better than on the configuration with randomly scattered red circles, suggesting it is able to use the alignment information to reach above-chance accuracy, but not in a completely reliable way. As a contrast, the RDS layer performs near-perfectly on this configuration, showing that the additional connectivity of the RDS helps it in discovering exploitable regularities in the data.

B.6 Training on less examples

In this section we vary the number of unique examples presented to the models in the training set. We keep the same number of optimizer steps as in the main experiments, but we reduce the number of samples we train on. The results for Identification are presented in Table 3, and the results for Discrimination are reported in Table 4.

In both Tables, in the first two rows we see all models overfitting the dataset, their test accuracy being at 0.5. They are unable to transfer to the training set and performing at chance levels. Then, respectively at 1000 samples for Identification and at 10000 samples for Discrimination the performance levels rise very close to their final levels. We wanted to observe whether the additional relational inductive biases in MPGNNs would allow for faster training than RDS and Deep Set; however we do not observe this: all models seem to have similar progression levels as the size of the training set increases. From this we conclude that the advantage of MPGNNs do not stem from their sample-efficiency, but rather from their ability to represent more complex functions.

	MPGNN	RDS	Deep Set
10	0.52 ± 0.038	0.52 ± 0.035	0.52 ± 0.032
100	**0.64** ± 0.051	0.58 ± 0.035	0.54 ± 0.019
1000	**0.94** ± 0.041	0.86 ± 0.065	0.61 ± 0.036
10000	**0.97** ± 0.026	0.91 ± 0.062	0.65 ± 0.079

Table B.2: Mean accuracies for training on reduced numbers of examples on Identification. The last row represents the full training set.

B.7 Adding Distractor Objects

In realistic environments cluttered with objects, only some of the objects could be relevant for the similarity task at hand; some of the objects may be distractors unrelated to the task. To test how robust our models are to additional objects in the input that bear no relevance to the task, we generate

	MPGNN	RDS	Deep Set
100	0.50 ± 0.005	0.50 ± 0.004	0.50 ± 0.005
1000	0.50 ± 0.004	0.50 ± 0.003	0.50 ± 0.005
10k	**0.87** ± 0.016	0.82 ± 0.098	0.52 ± 0.01
100k	**0.89** ± 0.03	0.80 ± 0.133	0.51 ± 0.014

Table B.3: Mean accuracies for training on reduced numbers of examples on Discrimination. The last row represents the full training set.

additional train and test sets for $n_{obj} \in [3..8]$. We use numbers of distractors $n_d \in [0..3]$ for both Identification and Discrimination. The results are reported in Table B.4.

Table B.4: Test accuracies on the distractor datasets.

Model	Identificatiom	Discrimination
MPGNN	**0.87** ± 0.043	**0.76** ± 0.019
RDS	0.78 ± 0.102	0.59 ± 0.069
Deep Set	0.67 ± 0.073	0.51 ± 0.01

We see the model performance consistently drop for MPGNN and RDS, with a decrease in test accuracy of around 10% on both tasks. The distractors seem to have no effect on Deep Set performance, suggesting that Deep Sets do not rely on a precise representation of object configuration. Dealing effectively with distractor objects could be done by adding an attention mechanism to the GNNs, a topic we leave for further work.

Appendix C

Appendix for experiments on grounding spatio-temporal language

C.1 Temporal Playground Specifications

C.1.1 Environment

Temporal Playground is a procedurally generated environment consisting of 3 objects and an agent's body. There are 32 types of objects, listed in Fig. C.1 along with 5 object categories. Each object has a continuous 2D position, a size, a continuous color code specified by a 3D vector in RGB space, a type specified by a one-hot vector, and a boolean unit specifying whether it is grasped. Note that categories are not encoded in the objects' features. The agent's body has its 2D position in the environment and its gripper state (grasping or non-grasping) as features. The size of the body feature vector is 3 while the object feature vector has a size of 39. This environment is a spatio-temporal extension of the one used in this work (Colas et al., 2020a).

All positions are constrained within $[-1, 1]^2$. The initial position of the agent is $(0, 0)$ while the initial object positions are randomized so that they are not in contact ($d(obj_1, obj_2) > 0.3$). Object sizes are sampled uniformly in $[0.2, 0.3]$, the size of the agent is 0.05. Objects can be grasped when the agent has nothing in hand, when it is close enough to the object center ($d(\text{agent}, obj) < (size(\text{agent}) + size(obj))/2$) and the gripper is closed (1, -1 when open). When a supply is on an animal or water is on a plant (contact define as distance between object being equal to the mean size of the two objects $d = (size(obj_1) + size(obj_2))/2$), the object will grow over time with a constant growth rate until it reaches the maximum size allowed for objects or until contact is lost.

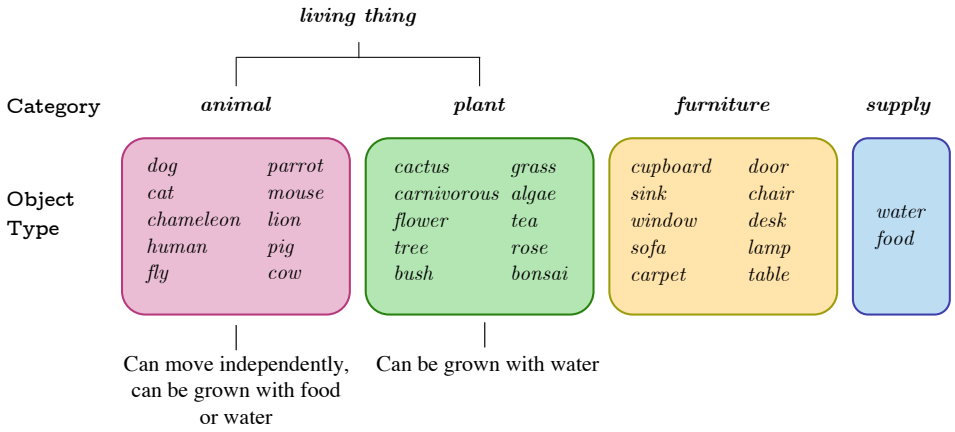


Figure C.1: **Representation of possible objects types and categories**. Information about the possible interactions between objects are also given.

C.1.2 Language

Grammar. The synthetic language we use can be decomposed into two components: the instantaneous grammar and the temporal logic. Both are specified through the BNF given in Figure C.2.

Instantaneous grammar:

```
<S>             ::= <pred> <thing_A>
<pred>          ::= grow | grasp | shake
<thing_A>       ::= <thing_B> | <attr> <thing_B> | thing <localizer> |
                    thing <localizer_all>
<localizer>     ::= left of <thing_B> | right of <thing_B> |
                    bottom of <thing_B> | top of <thing_B>
<localizer_all> ::= left most | right most | bottom most | top most
<thing_B>       ::= dog | cat | ... | thing
<attr>          ::= blue | green | red
```

Temporal aspect:

```
<S>             ::= was <pred> <thing_A>
<thing_A>       ::= thing was <localizer> | thing was <localizer_all>
```

Figure C.2: **BNF of the grammar used in Temporal Playground**. The instantaneous grammar allows generating true sentences about predicates, spatial relations (one-to-one and one to all). These sentences are then processed by the temporal logic to produce the linguistic descriptions of our observations; this step is illustrated in the Temporal Aspect rules. See the main text for information on how these sentences are generated.

Concept Definition. We split the set of all possible descriptions output by our grammar into four conceptual categories according to the rules given in Table C.1.

Concept	BNF	Size
1. Basic	<S> ::= <pred> <thing_A> <pred> ::= *grasp* <thing_A> ::= <thing_B> | <attr> <thing_B	152
2. Spatial	<S> ::= <pred> <thing_A> <pred> ::= *grasp* <thing_A> ::= *<thing* <localizer> | *thing* <localizer_all>	156
3. Temporal	<S> ::= <pred_A> <thing_A> | *was* <pred_B> <thing_A> <pred_A> ::= *grow* | *shake* <pred_B> ::= *grasp* | *grow* | *shake* <thing_A> ::= <thing_B> | <attr> <thing_B>	648
4. Spatio-Temporal	<S> ::= <pred_A> <thing_A> | *was* <pred_B> <thing_A> <pred_C> <thing_C> <pred_A> ::= *grow* | *shake* <pred_B> ::= *grasp* | *grow* | *shake* <pred_C> ::= *grasp* <thing_A> ::= *thing* <localizer> | *thing* <localizer_all> | *thing was* <localizer> | *thing was* <localizer_all> | <thing_C> ::= *thing was* <localizer> | *thing was* <localizer_all>	1716

Table C.1: **Concept categories with their associated BNF.** <thing_B>, <attr>, <localizer> and <localizer_all> are given in Fig. C.2

C.2 Supplementary Methods

C.2.1 Data Generation

Scripted bot. To generate the traces matching the descriptions of our grammar we define a set of scenarii that correspond to sequences of actions required to fulfill the predicates of our grammar, namely *grasp*, *grow* and *shake*. Those scenarii are then conditioned on a boolean that modulates them to obtain a mix of predicates in the present and the past tenses. For instance, if a *grasp* scenario is sampled, there will be a 50% chance that the scenario will end with the object being grasped, leading to a present-tense description; and a 50% chance that the agent releases the object, yielding a past tense description.

Description generation from behavioral traces of the agent. For each time step, the instantaneous grammar generates the set of all true instantaneous sentences using a set of filtering operations similar to the one used in CLEVR (Johnson et al., 2016), without the past predicates and past spatial relations. Then the temporal logic component uses these linguistic traces in the following way: if a given sentence for a predicate is true in a past time step and false in the present time step, the prefix token *'was'* is prepended to the sentence; similarly, if a given spatial relation is observed in a previous time step and unobserved in the present, the prefix token *'was'* is prepended to the spatial relation.

C.2.2 Input Encoding

We present the input processing in Fig. C.3. At each time step t, the body feature vector b_t and the object features vector $o_{i,t}$, $i = 1, 2, 3$ are encoded using two single-layer neural networks whose output are of size h. Similarly, each of the words of the sentence describing the trace (represented as one-hot vectors) is encoded and projected in the dimension of size h. We concatenate to the vector obtained a modality token m that defines if the output belongs to the scene $(1, 0)$ or to the description $(0, 1)$. We then feed the resulting vectors to a positional encoding that modulates the vectors according to the time step in the trace for b_t and $o_{i,t}$, $i = 1, 2, 3$ and according to the position of the word in the description for w_l.

We call the encoded body features $\hat{b}_t$ and it corresponds to $\hat{S}_{0,t}$ of the input tensor of our model (see Fig. 3.4 in the Main document). Similarly, $\hat{o}_{i,t}$, $i = 1, 2, 3$ are the encoded object features corresponding to $\hat{S}_{i,t}$, $i = 1, 2, 3$. Finally $\hat{w}_l$ are the encoded words and the components of tensor $\hat{W}$.

We call h the hidden size of our models and recall that $|\hat{b}_t| = |\hat{o}_{i,t}| = |\hat{w}_l| = h + 2$. This parameter is varied during the hyper-parameter search.

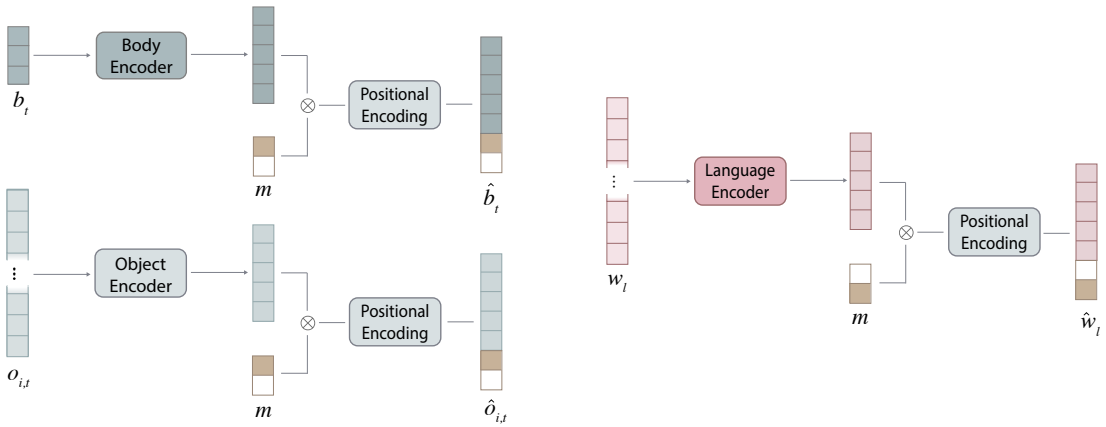


Figure C.3: **Input encoding.** Body, words and objects are all projected in the same dimension.

C.2.3 Details on LSTM models

To provide baseline models on our tasks we consider two LSTM variants. They are interesting baselines because they do not perform any relational computation except for relations between inputs at successive time steps. We consider the inputs as they were defined in Section 3.3.2 of the main paper. We consider two LSTM variants:

1. LSTM-FLAT: This variant has two internal LSTM: one that processes the language and one that processes the scenes as concatenations of all the body and object features. This produces two vectors that are concatenated into one, which is then run through an MLP and a final softmax to produce the final output.

2. LSTM-FACTORED: This variant independently processes the different body and object traces, which have previously been projected to the same dimension using a separate linear projection

for the object and for the body. The language is processed by a separate LSTM. These body, object and language vectors are finally concatenated and fed to a final MLP and a softmax to produce the output.

C.2.4 Details on Training Schedule

Implementation Details. The architectures are trained via backpropagation using the Adam Optimizer (Kingma & Ba, 2017). The data is fed to the model in batches of 512 examples for 150 000 steps. We use a modular buffer to sample an important variety of different descriptions in each batch and to impose a ratio of positive samples of 0.1 for each description in each batch.

Model implementations. We used the standard implementations of TransformerEncoderLayer and TransformerEncoder from pytorch version 1.7.1, as well as the default LSTM implementation. For initialization, we also use pytorch defaults.

Hyper-parameter search. To pick the best set of parameters for each of our eight models, we train them on 18 conditions and select the best models. Note that each condition is run for 3 seeds and best models are selected according to their averaged F_1 score on randomly held-out descriptions (15% of the sentences in each category given in Table C.1).

Best models. Best models obtained thanks to the parameter search are given in Table C.2.

Model	Learning rate	Model hyperparams			
		hidden size	layer count	head count	param count
UT	1e-4	256	4	8	1.3M
UT-WA	1e-5	512	4	8	14.0M
TFT	1e-4	256	4	4	3.5M
TFT-WA	1e-5	512	4	8	20.3M
SFT	1e-4	256	4	4	3.5M
SFT-WA	1e-4	256	2	8	2.7M
LSTM-FLAT	1e-4	512	4	N/A	15.6M
LSTM-FACTORED	1e-4	512	4	N/A	17.6M

Table C.2: **Hyperparameters.** (for all models)

Robustness to hyperparameters For some models, we have observed a lack of robustness to hyperparameters during our search. This translated to models learning to predict all observation-sentence tuples as false since the dataset is imbalanced (the proportion of true samples is 0.1). This behavior was systematically observed with a series of models whose hyperparameters are listed in Table C.3. This happens with the biggest models with high learning rates, especially with the -WA variants.

Model	Learning rate	Model hyperparams		
		hidden size	layer count	head count
UT-WA	1e-4	512	4	4
UT-WA	1e-4	512	4	8
SFT	1e-4	512	4	4
SFT-WA	1e-4	512	4	8
SFT-WA	1e-4	512	2	4
SFT-WA	1e-4	512	4	4
TFT	1e-4	512	4	4
TFT-WA	1e-4	512	4	8
TFT-WA	1e-4	512	2	4
TFT-WA	1e-4	512	4	4

Table C.3: **Unstable models.** Models and hyperparameters collapsing into uniform false prediction.

C.3 Supplementary Discussion: Formal descriptions of spatio-temporal meaning

The study of spatial and temporal aspects of language has a long history in Artificial Intelligence and linguistics, where researchers have tried to define formally the semantics of such uses of language. For instance, work in temporal logic (Allen, 1984) has tried to create rigorous definitions of various temporal aspects of action reflected in the English language, such as logical operations on time intervals (an action fulfilling itself simultaneously with another, before, or after), non-action events (standing still for one hour), and event causality. These formal approaches have been complemented by work in pragmatics trying to define language user's semantics as relates to spatial and temporal aspects of language. For instance, Tenbrink (2008) examines the possible analogies to be made between relationships between objects in the spatial domain and relationships between events in a temporal domain, and concludes empirically that these aspects of language are not isomorphic and have their own specific rules. Within the same perspective, a formal ontology of space is developed in (Bateman et al., 2010), whose complete system can be used to achieve contextualized interpretations of language users' spatial language. Spatial relations in everyday language use are also specified by the perspective used by the speaker; a formal account of this system is given in (Tenbrink, 2011), where the transferability of these representations to temporal relations between events is also studied. These lines of work are of great relevance to our approach, especially the ones involving spatial relationships. We circumvent the problem of reference frames by placing ourselves in an absolute reference system where the x-y directions unambiguously define the concepts of *left*, *right*, *top*, *bottom*; nevertheless these studies would be very useful in a context where the speaker would also be embodied and speak from a different perspective. As for the temporal aspect, these lines of work focus on temporal relations between separate events, which is not the object of our study here; we are concerned about single actions (as opposed to several events) unfolding, in the past or present, over several time steps.

C.4 Supplementary Results

C.4.1 Generalization to new observations from known sentences

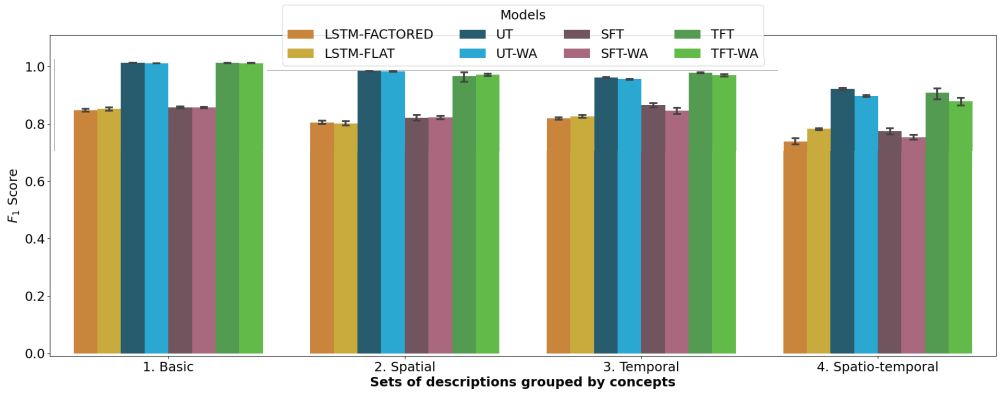


Figure C.4: **Generalization to new traces of observations.** F1 scores of all models on the train sentences with new observations. UT and TFT outperform other models on all four categories of meanings.

Appendix D

Appendix for ACES

D.1 Prompts

Here are the various prompts we use for ACES and all baselines.

D.1.1 Skills description.

Skills description used to label problem. see prompt D.1.2

Skills description

0 - **Sorting and Searching**: Sorting refers to arranging a collection of elements in a specific order, typically in ascending or descending order. Searching involves finding the location or presence of a particular element in a collection.
1 - **Counting and Combinatorics**: Understanding principles of counting and combinatorial analysis, including permutations, combinations, and other counting techniques. These skills are essential for solving problems that involve counting the number of possibilities or arrangements.
2 - **Trees and Graphs**: Analyzing and solving problems related to tree and graph structures involving nodes connected by edges. This includes tasks such as traversal, finding shortest paths, detecting cycles, and determining connectivity between nodes.
3 - **Mathematical Foundations**: Strong understanding of mathematical concepts such as summations, probability, arithmetics, and matrices.
4 - **Bit Manipulation**: Performing operations at the bit level to solve problems.
5 - **String Manipulation**: Operations and algorithms specifically designed for working with strings. This includes tasks like concatenation, searching, replacing, and parsing strings.
6 - **Geometry and Grid Problems**: Understanding geometric concepts and algorithms for problem-solving, including grid-related problems. This involves tasks such as grid traversal, finding distances, detecting patterns, and solving geometric problems on grids.
mybox 7 - **Recursion and Dynamic Programming**: Utilizing recursive techniques and dynamic programming approaches to solve problems by breaking them down into smaller subproblems and building solutions incrementally.
8 - **Stacks and Queues**: Data structures used to store and retrieve elements in a specific order. Stacks follow Last-In-First-Out, while queues follow First-In-First-Out. They are used for managing function calls, recursion, and implementing search algorithms.
9 - **Optimization Algorithms**: These algorithms aim to find the best possible solution

for a given problem by minimizing or maximizing an objective function. They involve searching for optimal values within a given solution space, considering various constraints and parameters. For example, brute-force search (checks all possible solutions to a problem without using heuristics) and greedy search (locally optimal choices at each step to find the best solution) are examples of optimization algorithms in this category.

D.1.2 Prompt for the puzzle labeler.

This prompt is used for ACES and all variants involving semantic representations.

Puzzle labelling

User: I will give you a Python programming puzzle f (and its solution g) and a list of programming skills. Your role is to say which programming skills are required to understand and solve the problem.
skills:

[Skills description]

The puzzle is:

[Insert Puzzle to label here]

After completing your reasoning (you can start by explaining the problem and the solution in a few sentences). Ensure you remove every listed skills that are unnecessary for understanding or solving the given problem. It is necessary to summarize your answer by writing every index of categories explicitly used in the problem or solution in a Python list, following the format provided below. Please ensure the correct usage of the following text where <Python list> is a list with numbers from 0 to 9: "Therefore, the list of indices for the problem is: <Python list>"

ChatGPT:

Example of puzzle labelling

Puzzle to label:

```
def f(s: str) -> bool:\\
    """Find a string with 1000 'o's but no two adjacent 'o'
       s."""\\
    return s.count('o') == 1000 and s.count('oo') == 0\\
def g():\\
    return ('h' + 'o') * 1000\\
assert f(g())\\
```

ChatGPT: To solve the problem, we need to understand the logic of the ‘f‘ function and how it checks for the presence of 1000 ’o’s and no adjacent ’o’s. We also need to understand the implementation of the ‘g‘ function, which generates the string with 1000 ’o’s.

Based on this understanding, the necessary programming skills are:
- 1: Counting and combinatorics (to understand the counting of ’o’s and ’oo’s in the string)
- 5: String Manipulation (to understand the string concatenation and searching)

Therefore, the list of indices for the problem is: [1, 5]

D.1.3 Prompt for the puzzle generator of ACES.

This prompt is used for ACES and all autotelic variants.

ACES

User:
I will give you 3 (Puzzle 0 to Puzzle 2) Python Programming Puzzle (P3). A P3 consists of a problem f and its corresponding solution g. The puzzle is solved if f(g()) == True. Your role is to generate new puzzles according to the instructions given.
In addition each of those puzzles are associated with a list of skills. Here is a detailed description of those skills:

[Skills description]

Your role is to generate 3 new puzzles (Puzzle 3 to Puzzle 5) that require those skills: **[Insert list skills to target]**.
Note that the first argument of f is the output g(). Make sure to define and set values for all arguments of the function ’f’ (excluding the first argument, as it is the solution that needs to be found and given by g). Both functions, ’f’ and ’g’ should have matching argument signatures: def f(arg0, arg1=value1, arg2=value2, ...) and def g(arg1=value1, arg2=value2, ...). Please provide all values (value1, value2, ...) for

all arguments. For example f(solution,arg1=1, arg2=2, ...) and g(arg1=1, arg2=2, ...). And you should not use f inside g.
Additionally, make sure to import any necessary libraries to ensure your code runs smoothly. Please ensure the mutated puzzles fall into all those skills: **[Insert list skills to target]**.
—-
Puzzle 0, required skills **[Insert list of skills associated with Puzzle 0]**:
[Insert Puzzle 0]
—
Puzzle 1, required skills **[Insert list of skills associated with Puzzle 1]** :
[Insert Puzzle 1]
—
Puzzle 2, required skills **[Insert list of skills associated with Puzzle 2]**:
[Insert Puzzle 2]
—

Could you please write 3 new interesting correct Python Programming Puzzles (from Puzzle 3 to Puzzle 5)? Please, ensure the new puzzles must necessitate the utilization of the following skills (required skills **[Insert list skills to target]**):
[index skill 1 to target - Name of the skill 1 targeted
[index skill 2 to target - Name of the skill 2 targeted]
.
.
.

ChatGPT:

D.1.4 Prompt for the puzzle generator of Static gen.

Static gen

User: I will give you 3 (Puzzle 0 to Puzzle 2) Python Programming Puzzle (P3). A P3 consists of a problem f and its corresponding solution g. The puzzle is solved if f(g()) == True. Your role is to write 3 new puzzles (Puzzle 3 to Puzzle 5). Note that the first argument of f is the output g(). Make sure to define and set values for all arguments of the function 'f' (excluding the first argument, as it is the solution that needs to be found and given by g).
Both functions, 'f' and 'g' should have matching argument signatures: def f(arg0, arg1=value1, arg2=value2, ...) and def g(arg1=value1, arg2=value2, ...). Please provide all values (value1, value2, ...) for all arguments. For example f(solution,arg1=1, arg2=2, ...) and g(arg1=1, arg2=2, ...). And you should not use f inside g.
Additionally, make sure to import any necessary libraries to ensure your code runs smoothly.
—-

Puzzle 0:

[Insert Puzzle]
—
Puzzle 1:

[Insert Puzzle]
—
Puzzle 2:

[Insert Puzzle]
—
ChatGPT:

D.1.5 Prompt for the puzzle generator of ELM and ELM semantic.

This prompt is used for non-autotelic baselines.

ELM and ELM semantic

User: I will give you 3 (Puzzle 0 to Puzzle 2) Python Programming Puzzle (P3). A P3 consists of a problem f and its corresponding solution g. The puzzle is solved if f(g()) == True. Your role is to write 3 new puzzles (Puzzle 3 to Puzzle 5). Note that the first argument of f is the output g(). Make sure to define and set values for all arguments of the function 'f' (excluding the first argument, as it is the solution that needs to be found and given by g).
Both functions, 'f' and 'g' should have matching argument signatures: def f(arg0, arg1=value1, arg2=value2, ...) and def g(arg1=value1, arg2=value2, ...). Please provide all values (value1, value2, ...) for all arguments. For example f(solution,arg1=1, arg2=2, ...) and g(arg1=1, arg2=2, ...). And you should not use f inside g.
Additionally, make sure to import any necessary libraries to ensure your code runs smoothly.
—-
Puzzle 0:

[Insert Puzzle]
—
Puzzle 1:

[Insert Puzzle]
—
Here is the puzzle to mutate:
Puzzle 2:

[Insert Puzzle to mutate]

—

Could you please mutate the Puzzle 2 into 3 new correct Python Programming Puzzles (from Puzzle 3 to Puzzle 5)? Please, ensure the mutated puzzles are meaningfully different from the existing puzzles.

ChatGPT:

Example Generation

Puzzle to mutate:

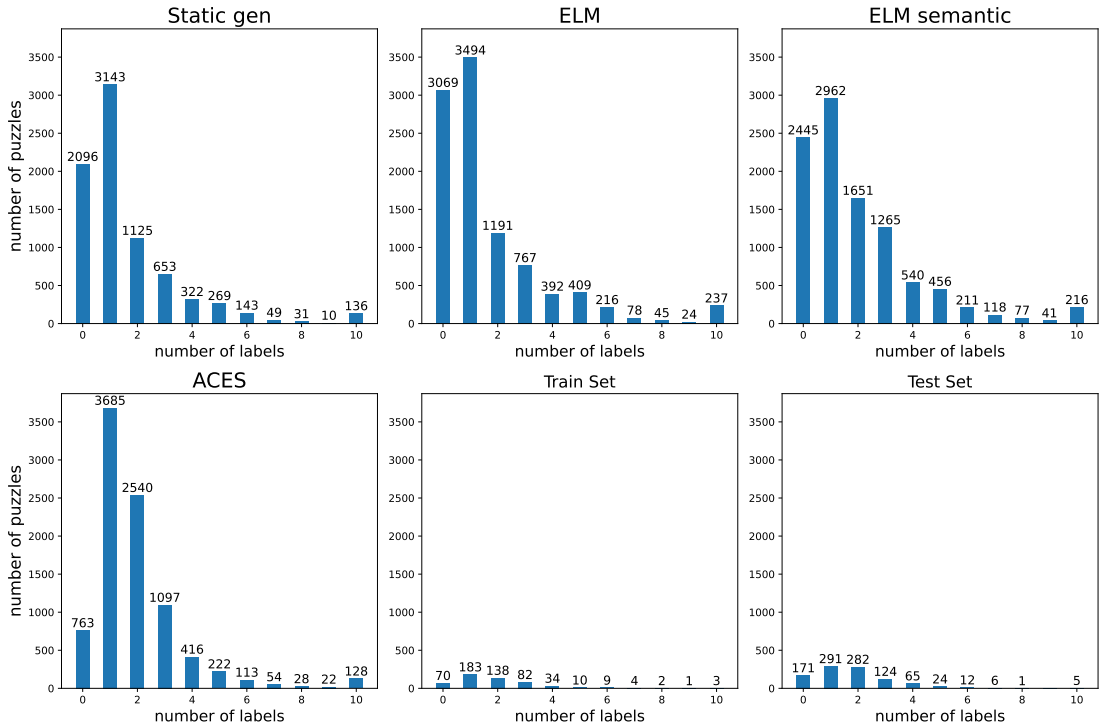
```
from typing import*
def f(n: int, lst=['apple', 'banana', 'orange', 'grape'])
    -> bool:
    """Check if the given element n is a prefix of any
       element in the list lst"""
    for word in lst:
        if word.startswith(n):
            return True
    return False

def g(lst=['apple', 'banana', 'orange', 'grape']):
    return lst[1]

assert f(g())
```

ChatGPT:

```
from typing import*
import math
def f(n: int, lst=[1, 2, 3, 4, 5]) -> bool:
    """Check if the given list contains any perfect square
       number and if it is divisible by n."""
    for num in lst:
        if math.isqrt(num)**2 == num:
            return n % num == 0
    return False
def g(lst=[1, 2, 3, 4, 5]):
    for num in lst:
        if math.isqrt(num)**2 == num:
            return num
    return None
assert f(g())
```

D.2 Sampling examples for confusion matrix computation

We sample 60 puzzles from one of our ACES experiments that we manually label with the sat of skills. To make sure all skills are represented in the sampled puzzles while ensuring we stay close enough to the distribution of puzzles encountered in training, we sample half 1 of these puzzles uniformly over all generated puzzles in the seed, and we sample half 2 according to the following principle: each time we sample a new puzzle we sample among the list of puzzles involving the currently least represented skill. While examining the dataset (see Figure D.1) we noticed a certain number of outlier puzzles which had been labeled with all the skills, and which would be overrepresented in the dataset because they contain all (and thus rare) skills. When inspecting these puzzles we did not find them to be a true combination of all skills. Thus when sampling half 2 (but not half 1) of the manually labeled puzzles we have excluded the anomalous puzzles labeled with all labels.



(a) random-gen

Figure D.1: Distributions of number of skills labeled by ChatGPT for (a) ACES, (b) ACES-random, (c) ELM-Semantic, (d) ELM, (e) static gen. A noteworthy effect of the goal-targeting in ACES and ACES-random is the low number of puzzle with no skill labels compared to the other methods. Goal-targetting seems to have an effect of how much generated puzzles fit to the predefined ontology.

D.3 Examples of generated puzzles

In this section we present a few puzzles and solution generated by our different methods and examine them qualitatively. In example D.3, the generated puzzles combines a string manipulation problem, a grid problem and a recursion problem, through the search for a path in a grid filled with

characters of a given input string. The example also illustrates the drift in task semantics mentioned in the main text: in this example the function `g` only gives the argument of the puzzle, which is both defined and solved in `f`. We additionally confirm our intuition of the drift in meaning of the puzzles through histograms of the proportion between puzzle and solution complexity, in Figure D.2. The intuition is that methods that generate puzzles with very short (and thus very simple) solutions have shifted from implementing algorithms in puzzles rather than in solutions. There is a clear difference between the train set from the P3 dataset and the data generation methods, with Static-Gen suffering less. This might have an influence on downstream performance.

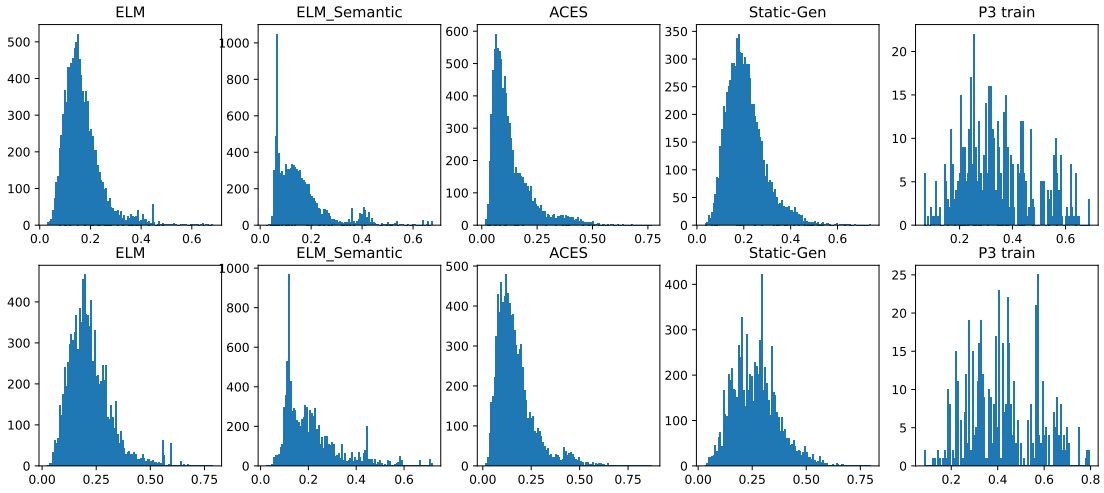


Figure D.2: Relative complexity of the generated solutions. We report the number of characters (top) and the size of nodes in the parsed abstract syntax tree (bottom) for the four algorithms and P3's train set (rightmost column). We can see a loss of relative complexity in all methods, less severe in Static-Gen.

Combination of String and Grid (ACES)

Mixture of a grid problem and of a character problem. Comments generated by the LLM inside the code, as well as the docstring, have been edited to shorten the figure.

```
from typing import*
from typing import List

def f(grid: List[List[int]], s: str) -> bool:
    def dfs(i: int, j: int, k: int) -> bool:
        if k == len(s):
            return True

        if grid[i][j] == s[k]:
            visited[i][j] = True

            for di, dj in [(1, 0), (-1, 0), (0, 1), (0, -1)
                ]:
                ni, nj = i + di, j + dj
                if 0 <= ni < len(grid) and 0 <= nj < len(
                    grid[0]) and not visited[ni][nj]:
                    if dfs(ni, nj, k + 1):
                        return True

            visited[i][j] = False

        return False

    for i in range(len(grid)):
        for j in range(len(grid[0])):
            visited = [[False] * len(grid[0]) for _ in
                range(len(grid))]
            if dfs(i, j, 0):
                return True

    return False

def g():
    grid = [
        ['a', 'b', 'c', 'e'],
        ['s', 'f', 'c', 's'],
        ['a', 'd', 'e', 'e']
    ]
    return (grid, "abcced")
```

Example generation for ACES

Label: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
from typing import*
from typing import List

def f(nums: List[int]) -> List[int]:
    """Given a list of integers, sort the list in non-
       decreasing order using the bubble sort algorithm.
    Return the sorted list.
    """
    def bubble_sort(nums):
        n = len(nums)
        for i in range(n):
            for j in range(0, n - i - 1):
                if nums[j] > nums[j + 1]:
                    nums[j], nums[j + 1] = nums[j + 1],
                        nums[j]
        return nums

    return bubble_sort(nums)

def g():
    nums = [9, 5, 2, 8, 1]
    return nums
```

Labels: [1, 1, 0, 0, 0, 1, 0, 0, 0, 0]

```
from typing import*

def f(s: str, n: int) -> bool:
    """Check if the given string is a palindrome and if the
        given number is divisible by 3 and 5."""
    is_palindrome = s == s[::-1]

    is_divisible_by_three_and_five = n % 3 == 0 and n % 5
       == 0

    return is_palindrome and is_divisible_by_three_and_five

def g():
    s = "deed"
    n = 30
    return s, n
```

Example generation for ELM-Semantic

Label: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
from typing import*

def f(numbers: List[int], target_sum=100) -> bool:
    """Check if there are three numbers in the list that
       add up to the target sum."""
    for i in range(len(numbers)):
        for j in range(i+1, len(numbers)):
            for k in range(j+1, len(numbers)):
                if numbers[i] + numbers[j] + numbers[k] ==
                   target_sum:
                    return True
    return False

def g(target_sum=100):
    return [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Labels: [0, 1, 0, 0, 0, 1, 0, 0, 0, 0]

```
from typing import*

def f(words: List[str], prefix='in') -> bool:
    """Check if any word from the list starts with the
       given prefix"""
    return any((w.startswith(prefix)) for w in words)

def g(prefix='in'):
    return ['input', 'information', 'innovation', 'great']
```

Example generations for ELM

Labels: [1, 1, 0, 0, 0, 1, 0, 1, 0, 0]

```
from typing import*
from typing import List

def f(names: List[str], name_length: int) -> bool:
    """Check if there exists a name in the given list that
       has the specified length"""
    for name in names:
        if len(name) == name_length:
            return True
    return False

def g(names=['John', 'Alice', 'Bob', 'Eve'], name_length=5)
    :
    return names, name_length
```

Labels: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
from typing import*

def f(arr: List[int], k=10) -> bool:
    """Find if there exists a pair of integers in the array
        whose sum is equal to k"""
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            if arr[i] + arr[j] == k:
                return True
    return False

def g(k=10):
    return [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example generations for Static-Gen

Labels: [1, 0, 0, 1, 0, 0, 0, 0, 0, 0]

```
from typing import*

def f(n: int, m=5) -> bool:
    """Check if n is divisible by any prime number less
       than m"""
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
       41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
    return any(n % prime == 0 for prime in primes if prime
       < m)

def g(m=5):
    return 100
```

Label: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
from typing import*

def f(x: int, lst1=[7, 5, 3, 1], lst2=[1, 3, 5, 7]) -> bool
   :
    """Check if two lists are equal when reversed"""
    return lst1[::-1] == lst2

def g(lst1=[7, 5, 3, 1], lst2=[1, 3, 5, 7]):
    return lst1[::-1]
```

D.4 Additional results

Additionally, we can visualize language diversity by projecting the semantic embeddings into 2D space via dimensionality reduction techniques like UMAP. We expect to measure diversity in a set of puzzles by looking at the coverage of the map. As the diversity increases, they should achieve both a wider spread of embeddings and potentially clearer cluster separation compared to baselines, showcasing greater linguistic diversity. of puzzles' embedding projected in 2d plane with UMAP.

Figure D.3: Explanation of the representation used in the Figure D.4

figures/code_autotelic/plot_niches_explained.

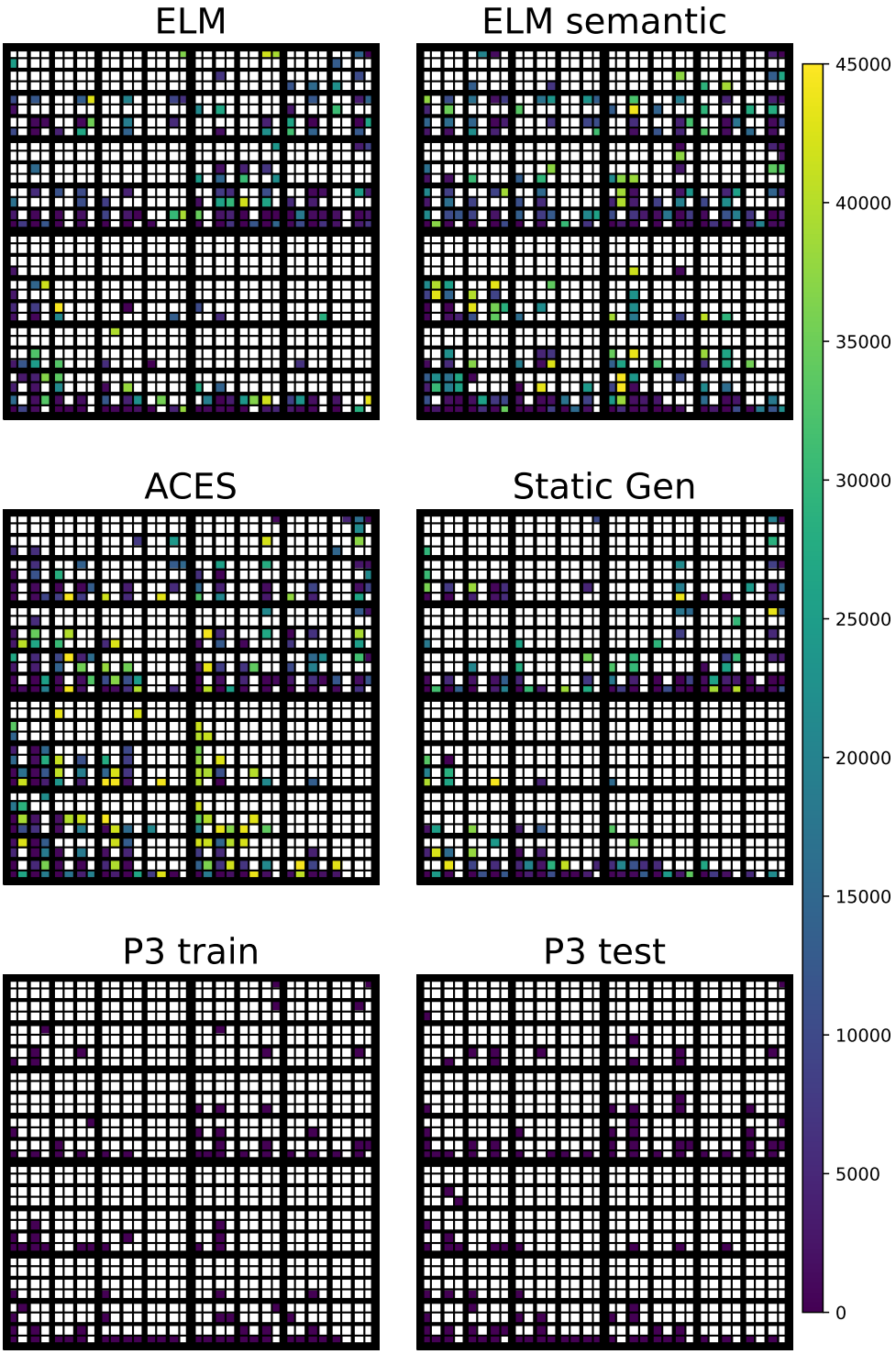


Figure D.4: Niche discovered over time 2d representation of cells filled in the skill space. More detail on the representation is given in the Figure D.3

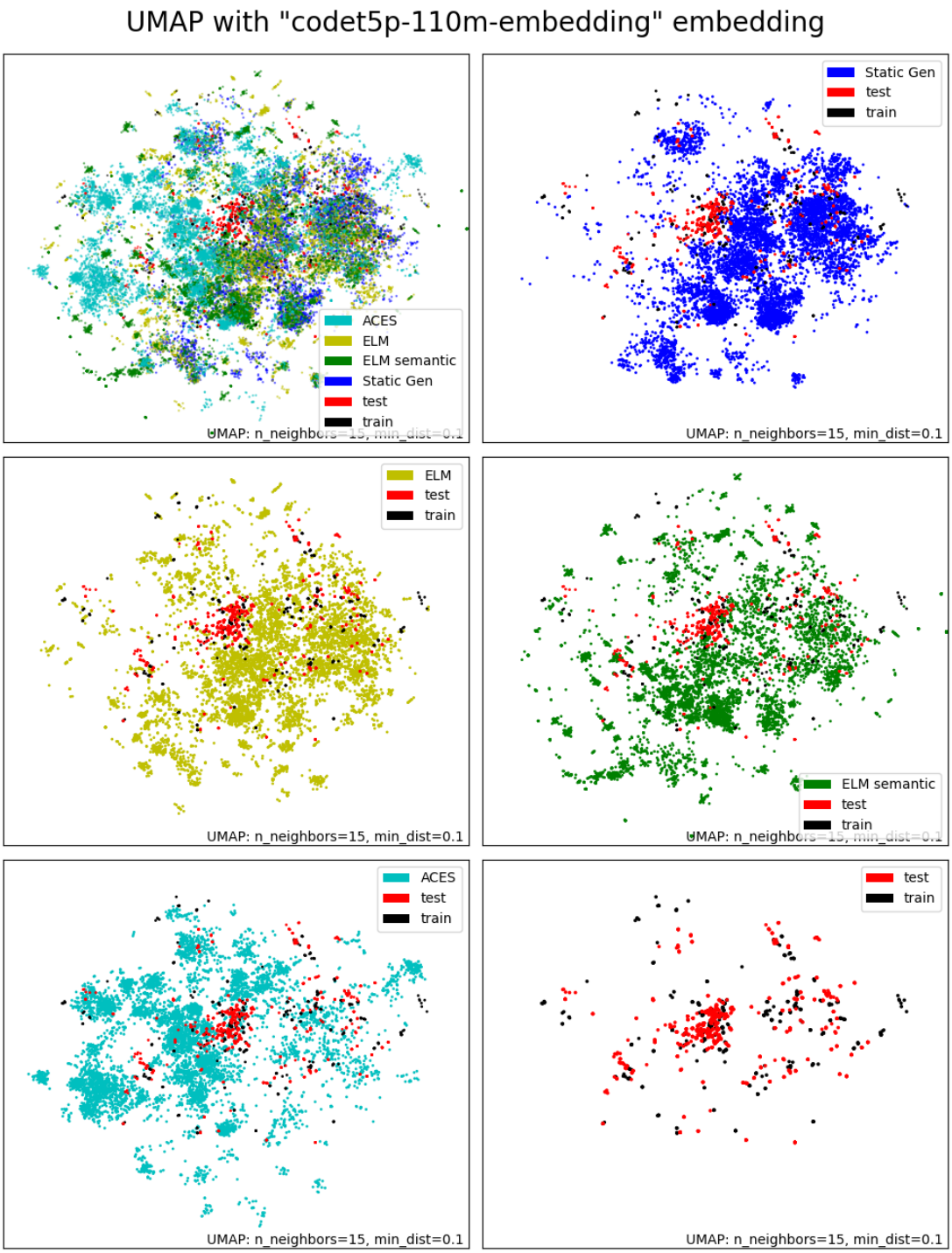


Figure D.5: UMAP projection of the Code5P-110M embeddings of discovered puzzles for one seed of each algorithm.

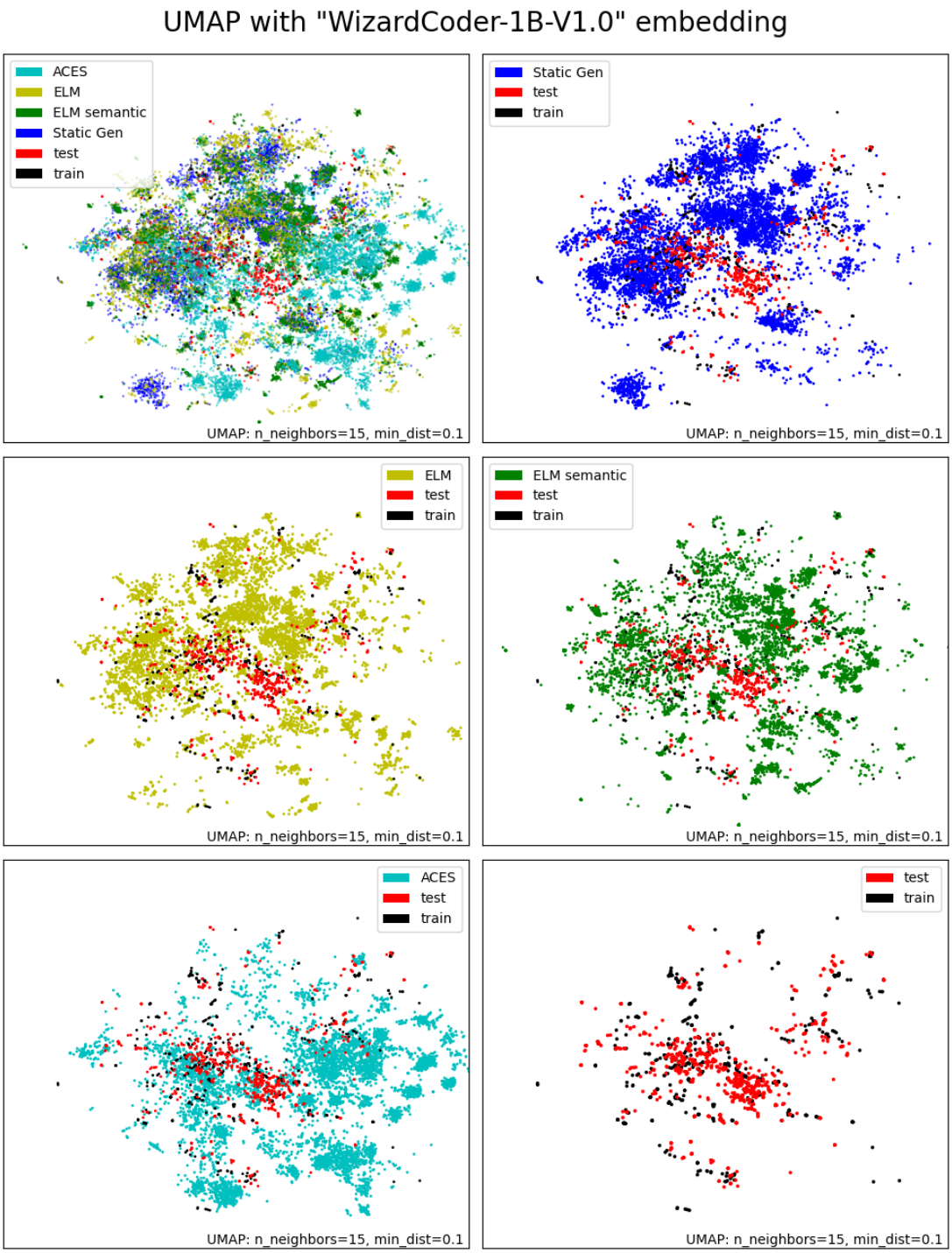


Figure D.6: UMAP projection of the WizardCoder-1B embeddings of discovered puzzles for one seed of each algorithm.

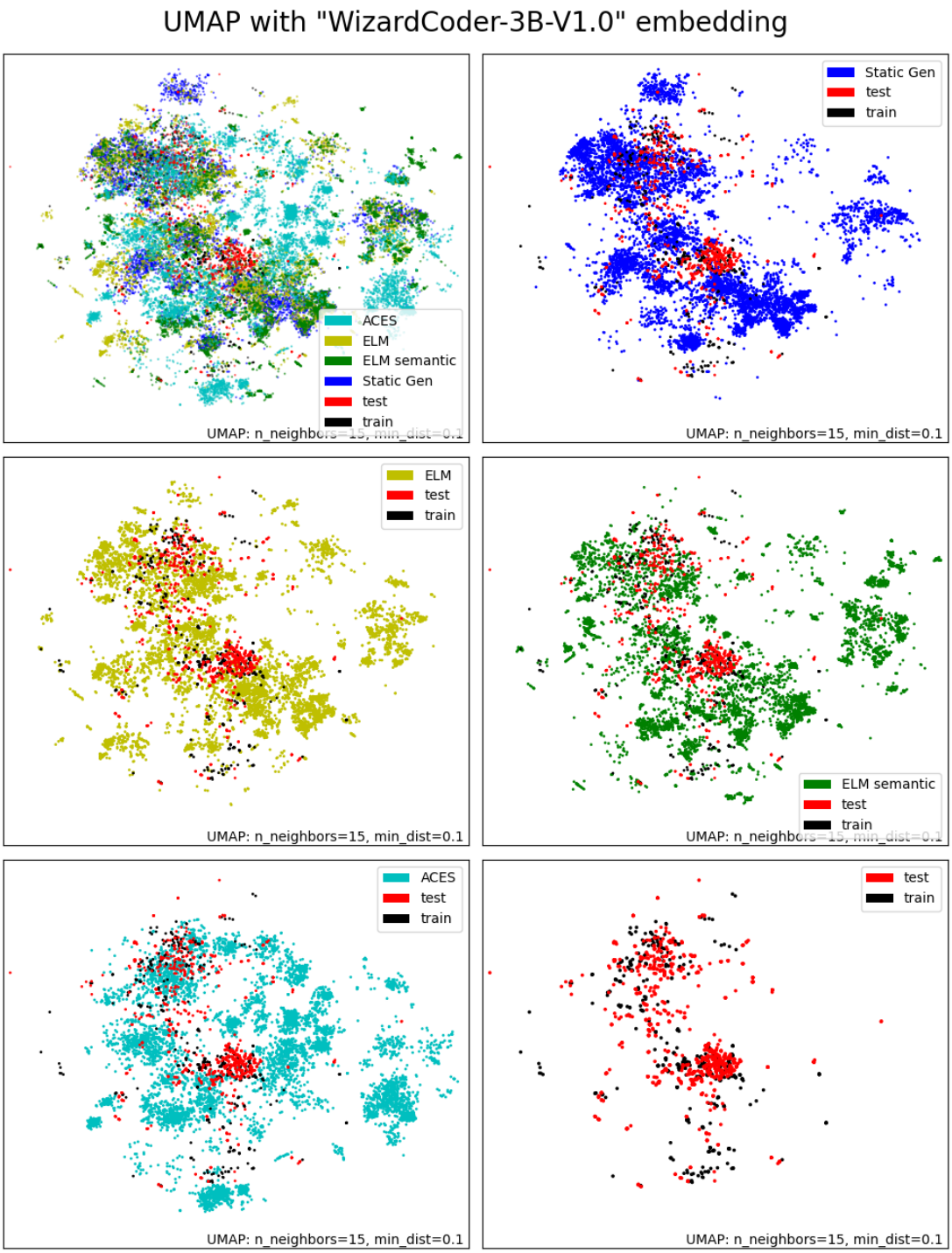


Figure D.7: UMAP projection of the WizardCoder-3B embeddings of discovered puzzles for one seed of each algorithm.

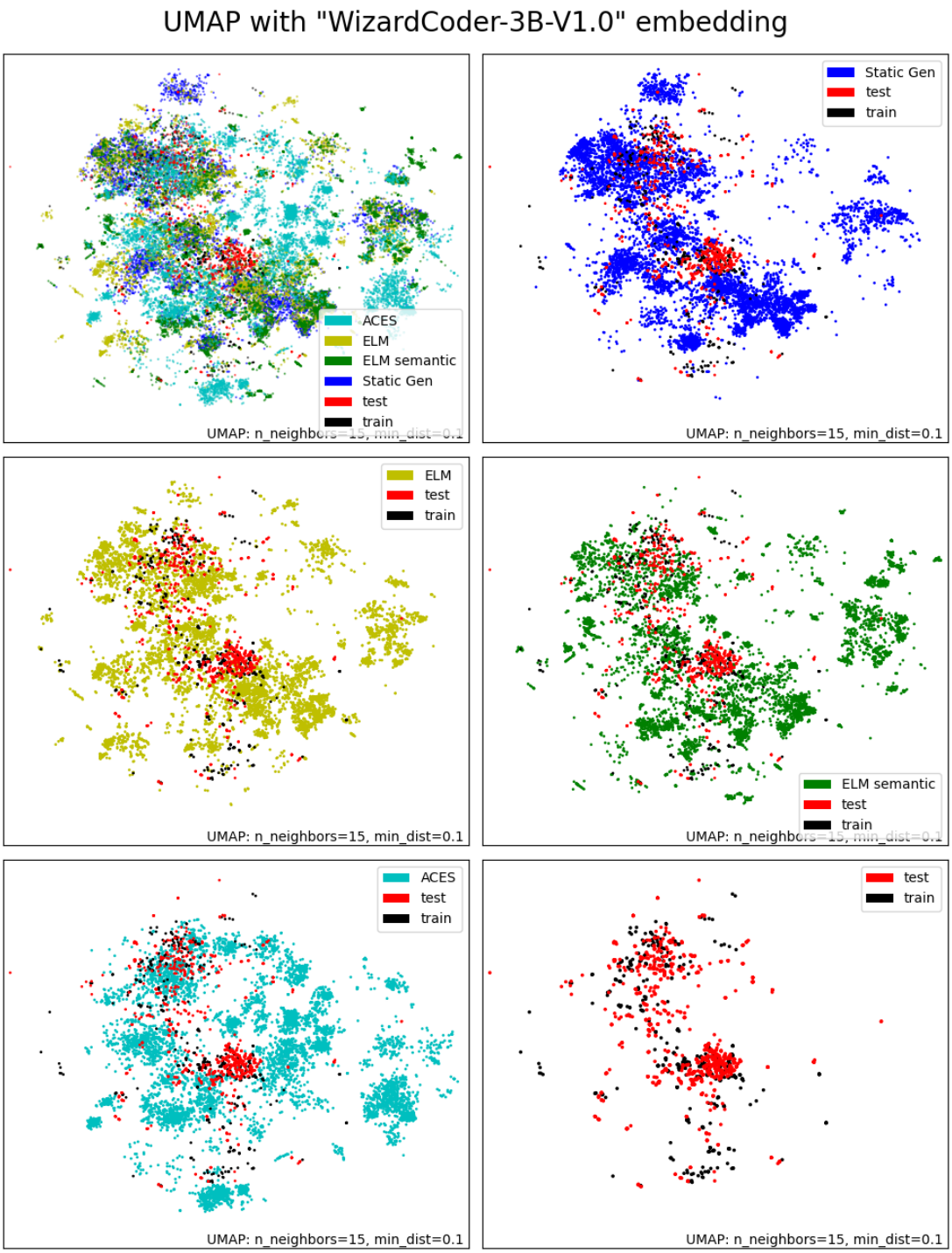


Figure D.8: UMAP projection of the WizardCoder-3B embeddings of discovered puzzles for one seed of each algorithm.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X.
TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
URL https://www.tensorflow.org/.
Software available from tensorflow.org.

Adelman, M. A.
Comment on the" h" concentration measure as a numbers-equivalent.
The Review of economics and statistics, pp. 99–101, 1969.

Ady, N. M., Shariff, R., Günther, J., and Pilarski, P. M.
Five properties of specific curiosity you didn't know curious machines should have, 2022.

Agassi, J. and Wiezenbaum, J.
Computer power and human reason: From judgment to calculation.
Technology and Culture, 17(4):813, October 1976.
doi: 10.2307/3103715.
URL https://doi.org/10.2307/3103715.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A.
Do as i can and not as i say: Grounding language in robotic affordances.
In *arXiv preprint arXiv:2204.01691*, 2022a.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., et al.
Do as i can, not as i say: Grounding language in robotic affordances.
arXiv preprint arXiv:2204.01691, 2022b.

Akakzia, A., Colas, C., Oudeyer, P.-Y., Chetouani, M., and Sigaud, O.
Grounding Language to Autonomously-Acquired Skills via Goal Generation.
In *ICLR 2021 - Ninth International Conference on Learning Representation*, Vienna / Virtual, Austria, May 2021.
URL https://hal.inria.fr/hal-03121146.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al.
Flamingo: a visual language model for few-shot learning.
Advances in Neural Information Processing Systems, 35:23716–23736, 2022.

Allen, J. F.
Towards a general theory of action and time.
Artificial Intelligence, 23(2):123–154, 1984.
ISSN 0004-3702.
doi: https://doi.org/10.1016/0004-3702(84)90008-0.
URL https://www.sciencedirect.com/science/article/pii/0004370284900080.

Altman, S.
Planning for agi and beyond.
Available at https://openai.com/blog/planning-for-agi-and-beyond (Accessed 2023/10/02), 2023.

Ammanabrolu, P., Tien, E., Luo, Z., and Riedl, M. O.
How to avoid being eaten by a grue: Exploration strategies for text-adventure agents.
arXiv preprint arXiv:2002.08795, 2020.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and Van Den Hengel, A.
Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3674–3683, 2018a.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A.
Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments.
In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3674–3683, 2018b.
doi: 10.1109/CVPR.2018.00387.

Andreas, J.
Measuring compositionality in representation learning.
arXiv preprint arXiv:1902.07181, 2019.

Andreas, J.
Language models as agent models.
arXiv preprint arXiv:2212.01681, 2022.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D.
Learning to compose neural networks for question answering.
arXiv preprint arXiv:1601.01705, 2016.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W.
Hindsight experience replay.
Advances in neural information processing systems, 30, 2017.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D.
VQA: Visual Question Answering.
In *International Conference on Computer Vision (ICCV)*, 2015.

Arora, S. and Doshi, P.
A survey of inverse reinforcement learning: Challenges, methods and progress.
Artificial Intelligence, 297:103500, 2021.

Arulkumaran, K., Cully, A., and Togelius, J.
Alphastar: An evolutionary computation perspective.
In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pp. 314–315, New York, NY, USA, 2019. Association for Computing Machinery.
ISBN 9781450367486.
doi: 10.1145/3319619.3321894.
URL https://doi.org/10.1145/3319619.3321894.

ASANO, T., KOJIMA, T., MATSUZAWA, T., KUBOTA, K., and MUROFUSHI, K.
Object and color naming in chimpanzees (pan troglodytes).
Proceedings of the Japan Academy, Series B, 58(5):118–122, 1982.
doi: 10.2183/pjab.58.118.
URL https://doi.org/10.2183/pjab.58.118.

Aubret, A., Matignon, L., and Hassas, S.
A survey on intrinsic motivation in reinforcement learning.
arXiv preprint arXiv:1908.06976, 2019.

Ba, J. L., Kiros, J. R., and Hinton, G. E.
Layer normalization.
arXiv preprint arXiv:1607.06450, 2016.

Backus, J. W.
The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference.
In *IFIP Congress*, 1959.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C.
Agent57: Outperforming the atari human benchmark.
In *International conference on machine learning*, pp. 507–517. PMLR, 2020.

Bahdanau, D., Cho, K., and Bengio, Y.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., and Courville, A.
Systematic generalization: what is required and can it be learned?
arXiv preprint arXiv:1811.12889, 2018.

Bahdanau, D., Hill, F., Leike, J., Hughes, E., Kohli, P., and Grefenstette, E.
Learning to understand goal specifications by modelling reward.
In *International Conference on Learning Representations*, 2019.
URL https://openreview.net/forum?id=H1xsSjC9Ym.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukosuite, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T., Hume, T., Bowman, S. R., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T., and Kaplan, J.
Constitutional AI: Harmlessness from AI Feedback, December 2022.
URL http://arxiv.org/abs/2212.08073.
arXiv:2212.08073 [cs].

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I.
Emergent tool use from multi-agent autocurricula.
arXiv preprint arXiv:1909.07528, 2019.

Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., Rowe, D., Shi, W., Spisak, J., Wei, A., Wu, D., Zhang, H., and Zijlstra, M.
Human-level play in the game of <idiplomacy</i by combining language models with strategic reasoning.
Science, 378(6624):1067–1074, December 2022.
doi: 10.1126/science.ade9097.
URL https://doi.org/10.1126/science.ade9097.

Baldassarre, G. and Mirolli, M. (eds.).
Intrinsically Motivated Learning in Natural and Artificial Systems.
Springer Berlin Heidelberg, November 2012.
doi: 10.1007/978-3-642-32375-1.
URL https://doi.org/10.1007/978-3-642-32375-1.

Baranes, A. and Oudeyer, P.-Y.
Active learning of inverse models with intrinsically motivated goal exploration in robots.
Robotics and Autonomous Systems, 61(1):49–73, jan 2013a.
doi: 10.1016/j.robot.2012.05.008.
URL https://doi.org/10.1016%2Fj.robot.2012.05.008.

Baranes, A. and Oudeyer, P.-Y.
Active learning of inverse models with intrinsically motivated goal exploration in robots.
Robotics and Autonomous Systems, 61(1):49–73, 2013b.

Baranes, A. and Oudeyer, P.-Y.
Active learning of inverse models with intrinsically motivated goal exploration in robots.
Robotics and Autonomous Systems, 61(1):49–73, January 2013c.
ISSN 0921-8890.
doi: 10.1016/j.robot.2012.05.008.
URL https://www.sciencedirect.com/science/article/pii/S0921889012000644.

Barto, A. G.
Intrinsic motivation and reinforcement learning.
In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47. Springer Berlin Heidelberg, nov 2012.
doi: 10.1007/978-3-642-32375-1_2.
URL https://doi.org/10.1007%2F978-3-642-32375-1_2.

Bateman, J. A., Hois, J., Ross, R., and Tenbrink, T.
A linguistic ontology of space for natural language processing.
Artificial Intelligence, 174(14):1027–1071, 2010.
ISSN 0004-3702.
doi: https://doi.org/10.1016/j.artint.2010.05.008.
URL https://www.sciencedirect.com/science/article/pii/S0004370210000858.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K.
Interaction networks for learning about objects, relations and physics.
CoRR, abs/1612.00222, 2016.
URL http://arxiv.org/abs/1612.00222.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R.
Relational inductive biases, deep learning, and graph networks, 2018a.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R.
Relational inductive biases, deep learning, and graph networks.
CoRR, abs/1806.01261, 2018b.
URL http://arxiv.org/abs/1806.01261.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R.
Unifying count-based exploration and intrinsic motivation.
Advances in neural information processing systems, 29, 2016a.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R.
Unifying count-based exploration and intrinsic motivation.
In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 1479–1487, Red Hook, NY, USA, 2016b. Curran Associates Inc.

ISBN 9781510838819.

Bellemare, M. G., Dabney, W., and Munos, R.
A distributional perspective on reinforcement learning.
In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

Bellman, R. E.
Dynamic programming.
Princeton Landmarks in Mathematics and Physics. Princeton University Press, Princeton, NJ, July 1957.

Bengio, Y., Ducharme, R., and Vincent, P.
A neural probabilistic language model.
10 2001.

Berlyne, D. E.
Conflict, arousal, and curiosity.
McGraw-Hill Book Company, 1960.
doi: 10.1037/11164-000.
URL https://doi.org/10.1037%2F11164-000.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.
Dota 2 with large scale deep reinforcement learning.
arXiv preprint arXiv:1912.06680, 2019.

Bishop, C. M.
Foreword.
In Saad, D. (ed.), *On-Line Learning in Neural Networks*, pp. 1–2. Cambridge University Press, Cambridge, January 1999.

Boden, M. A.
Chapter 9 - creativity.
In Boden, M. A. (ed.), *Artificial Intelligence*, Handbook of Perception and Cognition, pp. 267–291. Academic Press.
ISBN 978-0-12-161964-0.
doi: https://doi.org/10.1016/B978-012161964-0/50011-X.
URL https://www.sciencedirect.com/science/article/pii/B978012161964050011X.

Boden, M. A.
Creativity and artificial intelligence.
Artificial intelligence, 103(1-2):347–356, 1998.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al.
On the opportunities and risks of foundation models.
arXiv preprint arXiv:2108.07258, 2021.

Bostrom, N.
Superintelligence.
Oxford University Press, London, England, July 2014a.

Bostrom, N.
Superintelligence.
Oxford University Press, London, England, July 2014b.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q.
JAX: composable transformations of Python+NumPy programs, 2018.
URL http://github.com/google/jax.

Bradley, H., Dai, A., Zhang, J., Clune, J., Stanley, K., and Lehman, J.
Quality diversity through ai feedback.
CarperAI Blog, May 2023a.
URL https://carper.ai/quality-diversity-through-ai-feedback/.

Bradley, H., Fan, H., Carvalho, F., Fisher, M., Castricato, L., reciprocated, dmayhem93, Purohit, S., and Lehman, J.
OpenELM, January 2023b.
URL https://github.com/CarperAI/OpenELM.

Brakke, K. E. and Savage-Rumbaugh, E.
The development of language skills in bonobo and chimpanzee—i. comprehension.
Language &amp Communication, 15(2):121–148, April 1995.
doi: 10.1016/0271-5309(95)00001-7.
URL https://doi.org/10.1016/0271-5309(95)00001-7.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.
Language models are few-shot learners.
Advances in neural information processing systems, 33:1877–1901, 2020.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y.
Spectral networks and locally connected networks on graphs.
In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

Campero, A., Raileanu, R., Kuttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E.
Learning with AMIGo: Adversarially motivated intrinsic goals.
In *International Conference on Learning Representations*, 2021.
URL https://openreview.net/forum?id=ETBc_MIMgoX.

Cangelosi, A., Metta, G., Sagerer, G., Nolfi, S., Nehaniv, C., Fischer, K., Tani, J., Belpaeme, T., Sandini, G., Nori, F., Fadiga, L., Wrede, B., Rohlfing, K., Tuci, E., Dautenhahn, K., Saunders, J., and Zeschel, A.
Integration of action and language knowledge: A roadmap for developmental robotics.
IEEE Transactions on Autonomous Mental Development, 2(3):167–195, 2010.
doi: 10.1109/TAMD.2010.2053034.

Carey, S. and Bartlett, E. J.
Acquiring a single new word.
1978.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S.
End-to-end object detection with transformers, 2020.

Carroll, S. B.
Endless forms most beautiful.
WW Norton, New York, NY, April 2006a.

Carroll, S. B.
Endless forms most beautiful.
WW Norton, New York, NY, April 2006b.

Carruthers, P.
Thinking in language? Evolution and a modularist possibility, pp. 94–120.
Cambridge University Press, 1998.
doi: 10.1017/CBO9780511597909.007.

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y.
Grounding large language models in interactive environments with online reinforcement learning.
arXiv preprint arXiv:2302.02662, 2023.

Caucheteux, C. and King, J.-R.
Brains and algorithms partially converge in natural language processing.
Communications Biology, 5(1):134, Feb 2022.
ISSN 2399-3642.
doi: 10.1038/s42003-022-03036-1.
URL https://doi.org/10.1038/s42003-022-03036-1.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R.
Gated-attention architectures for task-oriented language grounding, 2018.

Charms, R. d.
Personal causation.
Lawrence Erlbaum Associates, Mahwah, NJ, August 1983.

Chen, D. L. and Mooney, R. J.
Learning to interpret natural language navigation instructions from observations.
In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pp. 859–865. AAAI Press, 2011.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I.
Decision transformer: Reinforcement learning via sequence modeling.
In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 15084–15097. Curran Associates, Inc., 2021a.
URL https://proceedings.neurips.cc/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I.
Generative pretraining from pixels.
In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1691–1703. PMLR, 13–18 Jul 2020a.
URL https://proceedings.mlr.press/v119/chen20s.html.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W.
Evaluating Large Language Models Trained on Code, July 2021b.
URL http://arxiv.org/abs/2107.03374.
arXiv:2107.03374 [cs].

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G.
A simple framework for contrastive learning of visual representations.
In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020b.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y.
Babyai: A platform to study the sample efficiency of grounded language learning, 2019.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P.
Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.
URL https://lmsys.org/blog/2023-03-30-vicuna/.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y.
Learning phrase representations using RNN encoder–decoder for statistical machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.
doi: 10.3115/v1/D14-1179.
URL https://aclanthology.org/D14-1179.

Chomsky, N.
A review of skinner's verbal behavior.
Language, 35(1):26, January 1959.
doi: 10.2307/411334.
URL https://doi.org/10.2307/411334.

Chomsky, N.
Aspects of the theory of syntax.
MIT Press, London, England, January 1965.

Chomsky, N.
The minimalist program.
The MIT Press. MIT Press, London, England, September 1995.

Chomsky, N. and Lightfoot, D. W.
Syntactic structures.
Walter de Gruyter, 2002.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al.
Palm: Scaling language modeling with pathways.
arXiv preprint arXiv:2204.02311, 2022.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D.
Deep reinforcement learning from human preferences.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a.
URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D.
Deep reinforcement learning from human preferences.
Advances in neural information processing systems, 30, 2017b.

Chu, J. and Schulz, L.
In praise of folly: Flexible goals and human cognition.
mar 2023a.
doi: 10.31234/osf.io/zxbqr.
URL https://doi.org/10.31234%2Fosf.io%2Fzxbqr.

Chu, J. and Schulz, L. E.
Play, curiosity, and cognition.
Annual Review of Developmental Psychology, 2:317–343, 2020.

Chu, J. and Schulz, L. E.
Not Playing by the Rules: Exploratory Play, Rational Action, and Efficient Search.
Open Mind, 7:294–317, 06 2023b.
ISSN 2470-2986.
doi: 10.1162/opmi_a_00076.
URL https://doi.org/10.1162/opmi_a_00076.

Church, A.
A set of postulates for the foundation of logic.
The Annals of Mathematics, 33(2):346, April 1932.
doi: 10.2307/1968337.
URL https://doi.org/10.2307/1968337.

Cideron, G., Seurin, M., Strub, F., and Pietquin, O.
Higher: Improving instruction following with hindsight generation for experience replay.

In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 225–232, 2020.
doi: 10.1109/SSCI47803.2020.9308603.

Cinque, G. and Rizzi, L.
The cartography of syntactic structures.
In *The Oxford Handbook of Linguistic Analysis*, pp. 51–66. Oxford University Press, September 2012.
doi: 10.1093/oxfordhb/9780199544004.013.0003.
URL https://doi.org/10.1093/oxfordhb/9780199544004.013.0003.

Clark, A.
Linguistic anchors in the sea of thought?
Pragmatics and Cognition, 4(1):93–103, January 1996.
doi: 10.1075/pc.4.1.09cla.
URL https://doi.org/10.1075/pc.4.1.09cla.

Clark, A.
Language, embodiment, and the cognitive niche.
Trends in Cognitive Sciences, 10(8):370–374, 2006.
doi: 10.1016/j.tics.2006.06.012.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D.
What does bert look at? an analysis of bert's attention.
arXiv preprint arXiv:1906.04341, 2019.

Clune, J.
Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence.
arXiv preprint arXiv:1905.10985, 2019.

Colas, C., Fournier, P., Chetouani, M., Sigaud, O., and Oudeyer, P.-Y.
Curious: intrinsically motivated modular multi-goal reinforcement learning.
In *International conference on machine learning*, pp. 1331–1340. PMLR, 2019a.

Colas, C., Fournier, P., Chetouani, M., Sigaud, O., and Oudeyer, P.-Y.
CURIOUS: Intrinsically motivated modular multi-goal reinforcement learning.
In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1331–1340. PMLR, 09–15 Jun 2019b.
URL https://proceedings.mlr.press/v97/colas19a.html.

Colas, C., Sigaud, O., and Oudeyer, P.-Y.
A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms, 2019c.

Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Moulin-Frier, C., Dominey, P., and Oudeyer, P.-Y.
Language as a cognitive tool to imagine goals in curiosity driven exploration.
In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3761–3774. Curran Associates, Inc., 2020a.
URL https://proceedings.neurips.cc/paper/2020/file/274e6fcf4a583de4a81c6376f17673e7-Paper.pdf.

Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Moulin-Frier, C., Dominey, P., and Oudeyer, P.-Y.
Language as a Cognitive Tool to Imagine Goals in Curiosity Driven Exploration.
In *Advances in Neural Information Processing Systems*, volume 33, pp. 3761–3774. Curran Associates, Inc., 2020b.
URL https://proceedings.neurips.cc/paper/2020/hash/274e6fcf4a583de4a81c6376f17673e7-Abstract.html.

Colas, C., Karch, T., Moulin-Frier, C., and Oudeyer, P.-Y.
Language as a Cognitive Tool: Dall-E, Humans and Vygotskian RL Agents, March 2021.
URL https://hal.archives-ouvertes.fr/hal-03159786.
This blog post presents a supra-communicative view of language and advocates for the use of language as a cognitive tool to organize the cognitive development of intrinsically motivated artificial agents. We go over studies revealing the cognitive functions of language in humans, cover similar uses of language in the design of artificial agents and advocate for the pursuit of Vygotskian embodied agents - artificial agents that leverage language as a cognitive tool to structure their continuous experience, form abstract representations, reason, imagine creative goals, plan towards them and simulate future possibilities.

Colas, C., Karch, T., Moulin-Frier, C., and Oudeyer, P.-Y.
Language and culture internalization for human-like autotelic ai.
Nature Machine Intelligence, 4(12):1068–1076, 2022a.

Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y.
Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey.
Journal of Artificial Intelligence Research, 74:1159–1199, 2022b.

Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y.
Autotelic Agents with Intrinsically Motivated Goal-Conditioned Reinforcement Learning: a Short Survey, July 2022c.
URL http://arxiv.org/abs/2012.09830.
arXiv:2012.09830 [cs].

Colas, C., Teodorescu, L., Oudeyer, P.-Y., Yuan, X., and Côté, M.-A.
Augmenting Autotelic Agents with Large Language Models.
arXiv preprint arXiv:2305.12487, 2023.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., El Asri, L., Adada, M., et al.
Textworld: A learning environment for text-based games.
In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pp. 41–75. Springer, 2019.

Creswell, A., Nikiforou, K., Vinyals, O., Saraiva, A., Kabra, R., Matthey, L., Burgess, C., Reynolds, M., Tanburn, R., Garnelo, M., and Shanahan, M.
Alignnet: Unsupervised entity alignment, 2020.

Creswell, A., Kabra, R., Burgess, C., and Shanahan, M.
Unsupervised object-based transition models for 3d partially observable environments.

CoRR, abs/2103.04693, 2021.
URL https://arxiv.org/abs/2103.04693.

Csikszentmihalyi, M.
Flow.
HarperCollins, Sydney, NSW, Australia, March 1990.

Csikszentmihalyi, M.
Finding Flow: The Psychology of Engagement With Everyday Life, pp. –144.
04 1998.
ISBN 0465024114.

Cully, A.
Autonomous skill discovery with quality-diversity and unsupervised descriptors.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89, 2019.

Cully, A. and Demiris, Y.
Quality and diversity optimization: A unifying modular framework.
IEEE Transactions on Evolutionary Computation, 22(2):245–259, 2018a.
doi: 10.1109/TEVC.2017.2704781.

Cully, A. and Demiris, Y.
Hierarchical behavioral repertoires with unsupervised descriptors.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 69–76, 2018b.

Dasgupta, I., Kaeser-Chen, C., Marino, K., Ahuja, A., Babayan, S., Hill, F., and Fergus, R.
Collaborating with language models for embodied reasoning.
arXiv preprint arXiv:2302.00763, 2023.

Davidson, G., Gureckis, T. M., and Lake, B. M.
Creativity, compositionality, and common sense in human goal generation.
mar 2022.
doi: 10.31234/osf.io/byzs5.
URL https://doi.org/10.31234%2Fosf.io%2Fbyzs5.

Dawkins, R.
The Selfish Gene.
Oxford University Press, London, England, October 1976.

Deacon, T.
The Symbolic Species, pp. 9–38.
01 2012.
ISBN 978-94-007-2335-1.
doi: 10.1007/978-94-007-2336-8_2.

Deci, E. L. and Ryan, R. M.
Intrinsic Motivation and Self-Determination in Human Behavior.
Springer US, 1985.
doi: 10.1007/978-1-4899-2271-7.
URL https://doi.org/10.1007%2F978-1-4899-2271-7.

Defferrard, M., Bresson, X., and Vandergheynst, P.
Convolutional neural networks on graphs with fast localized spectral filtering.
CoRR, abs/1606.09375, 2016.
URL http://arxiv.org/abs/1606.09375.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., and Riedmiller, M.
Magnetic control of tokamak plasmas through deep reinforcement learning.
Nature, 602(7897):414–419, February 2022.
doi: 10.1038/s41586-021-04301-9.
URL https://doi.org/10.1038/s41586-021-04301-9.

Dennett, D. C.
Consciousness explained.
Little, Brown, London, England, 1991.

Der, R. and Martius, G.
The Playful Machine.
Springer Berlin Heidelberg, 2012.
doi: 10.1007/978-3-642-20253-7.
URL https://doi.org/10.1007/978-3-642-20253-7.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.
Bert: Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv:1810.04805, 2018.

Ding, D., Hill, F., Santoro, A., and Botvinick, M.
Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures, 2020.

Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-Dickstein, J., et al.
Language model cascades.
arXiv preprint arXiv:2207.10342, 2022.

Donderi, D. C. and Zelnicker, D.
Parallel processing in visual same-different decisions.
Perception & Psychophysics, 5(4):197–200, Jul 1969.
ISSN 1532-5962.
doi: 10.3758/BF03210537.
URL https://doi.org/10.3758/BF03210537.

Dove, G.
Language as a disruptive technology: abstract concepts, embodiment and the flexible mind.
Philosophical Transactions of the Royal Society B: Biological Sciences, 373(1752):20170135, 2018.

Du, Y., Konyushkova, K., Denil, M., Raju, A., Landon, J., Hill, F., de Freitas, N., and Cabi, S.
Vision-language models as success detectors.
arXiv preprint arXiv:2303.07280, 2023a.

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J.
Guiding pretraining in reinforcement learning with large language models.
arXiv preprint arXiv:2302.06692, 2023b.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P.
Convolutional networks on graphs for learning molecular fingerprints.
In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2224–2232. Curran Associates, Inc., 2015.
URL http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J.
First return, then explore.
Nature, 590(7847):580–586, 2021.

Elman, J. L.
Finding structure in time.
Cognitive Science, 14(2):179–211, March 1990.
doi: 10.1207/s15516709cog1402_1.
URL https://doi.org/10.1207/s15516709cog1402_1.

Elman, J. L.
Learning and development in neural networks: the importance of starting small.
Cognition, 48(1):71–99, 1993.
ISSN 0010-0277.
doi: https://doi.org/10.1016/0010-0277(93)90058-4.
URL https://www.sciencedirect.com/science/article/pii/0010027793900584.

Elman, J. L.
Language as a dynamical system.
In van Gelder, T. and Port, R. (eds.), *Mind as Motion: Explorations in the Dynamics of Cognition*, pp. 195–223. MIT Press, 1995.

Engelcke, M., Kosiorek, A. R., Jones, O. P., and Posner, I.
Genesis: Generative scene inference and sampling with object-centric latent representations, 2020.

Enquist, M., Ghirlanda, S., Jarrick, A., and Wachtmeister, C.-A.
Why does human culture increase exponentially?
Theoretical Population Biology, 74(1):46–55, 2008.
ISSN 0040-5809.
doi: https://doi.org/10.1016/j.tpb.2008.04.007.
URL https://www.sciencedirect.com/science/article/pii/S004058090800052X.

Eslami, S. M. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., Wierstra, D., Kavukcuoglu, K., and Hassabis, D.
Neural scene representation and rendering.
Science, 360(6394):1204–1210, 2018.
doi: 10.1126/science.aar6170.
URL https://www.science.org/doi/abs/10.1126/science.aar6170.

Etcheverry, M., Moulin-Frier, C., and Oudeyer, P.-Y.
Hierarchically organized latent modules for exploratory search in morphogenetic systems.
Advances in Neural Information Processing Systems, 33:4846–4859, 2020.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S.
Diversity is all you need: Learning diverse skills without a reward function.
2018.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S.
Diversity is all you need: Learning skills without a reward function.
In *International Conference on Learning Representations*, 2019.
URL https://openreview.net/forum?id=SJx63jRqFm.

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A.
Minedojo: Building open-ended embodied agents with internet-scale knowledge.
Advances in Neural Information Processing Systems, 35:18343–18362, 2022.

Fillmore, C. J., Kay, P., and O'Connor, M. C.
Regularity and idiomaticity in grammatical constructions: The case of let alone.
Language, 64(3):501, September 1988.
doi: 10.2307/414531.
URL https://doi.org/10.2307/414531.

Fisher, R. A.
The genetical theory of natural selection.
Oxford University Press, London, England, October 1999.

Florensa, C., Held, D., Geng, X., and Abbeel, P.
Automatic goal generation for reinforcement learning agents.
In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

Fodor, J. A.
The language of thought.
The Language and Thought Series. Harvard University Press, London, England, July 1979.

Fogarty, L., Creanza, N., and Feldman, M. W.
Cultural evolutionary perspectives on creativity and human innovation.
Trends in Ecology &amp Evolution, 30(12):736–754, December 2015.
doi: 10.1016/j.tree.2015.10.004.
URL https://doi.org/10.1016/j.tree.2015.10.004.

Forestier, S. and Oudeyer, P.-Y.
Curiosity-Driven Development of Tool Use Precursors: a Computational Model.
In Papafragou, A., Grodner, D., Mirman, D., and Trueswell, J. (eds.), *38th Annual Conference of the Cognitive Science Society (CogSci 2016)*, , Proceedings of the 38th Annual Conference of the Cognitive Science Society, pp. 1859–1864, Philadelphie, PA, United States, August 2016.
URL https://hal.science/hal-01354013.

Forestier, S., Portelas, R., Mollard, Y., and Oudeyer, P.-Y.
Intrinsically motivated goal exploration processes with automatic curriculum learning.
Journal of Machine Learning Research, 2022a.

Forestier, S., Portelas, R., Mollard, Y., and Oudeyer, P.-Y.
Intrinsically motivated goal exploration processes with automatic curriculum learning.
The Journal of Machine Learning Research, 23(1):6818–6858, 2022b.
Publisher: JMLRORG.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al.
Noisy networks for exploration.
arXiv preprint arXiv:1706.10295, 2017.

French, R. M.
Catastrophic forgetting in connectionist networks.
Trends in Cognitive Sciences, 3(4):128–135, 1999.
ISSN 1364-6613.
doi: https://doi.org/10.1016/S1364-6613(99)01294-2.
URL https://www.sciencedirect.com/science/article/pii/S1364661399012942.

Friedman, D. and Dieng, A. B.
The vendi score: A diversity evaluation metric for machine learning.
arXiv preprint arXiv:2210.02410, 2022.

Fu, Y., Peng, H., Sabharwal, A., Clark, P., and Khot, T.
Complexity-based prompting for multi-step reasoning.
arXiv preprint arXiv:2210.00720, 2022.

Gardner, R. A. and Gardner, B. T.
Teaching sign language to a chimpanzee.
Science, 165(3894):664–672, August 1969.
doi: 10.1126/science.165.3894.664.
URL https://doi.org/10.1126/science.165.3894.664.

Geng, X. and Liu, H.
Openllama: An open reproduction of llama, May 2023.
URL https://github.com/openlm-research/open_llama.

Gentner, D.
Language as cognitive tool kit: How language supports relational thought.
American psychologist, 71(8):650, 2016.

Gentner, D. and Loewenstein, J.
Relational language and relational thought, 2002.

Gibson, J. J. J. J.
The senses considered as perceptual systems.
Allen & Unwin, London, 1968.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E.
Neural message passing for quantum chemistry.
CoRR, abs/1704.01212, 2017.
URL http://arxiv.org/abs/1704.01212.

Giurfa, M., Zhang, S., Jenett, A., Menzel, R., and Srinivasan, M. V.
The concepts of 'sameness' and 'difference' in an insect.
Nature, 410(6831):930–933, Apr 2001.
ISSN 1476-4687.
doi: 10.1038/35073582.
URL https://doi.org/10.1038/35073582.

Glorot, X. and Bengio, Y.
Understanding the difficulty of training deep feedforward neural networks.
In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.
URL https://proceedings.mlr.press/v9/glorot10a.html.

Gogate, L. J. and Bahrick, L. E.
Intersensory redundancy facilitates learning of arbitrary relations between vowel sounds and objects in seven-month-old infants.
J. Exp. Child Psychol., 69(2):133–149, May 1998.

Goldberg, A.
Constructions.
Cognitive Theory of Language & Culture S. University of Chicago Press, Chicago, IL, January 1995.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y.
Generative adversarial nets.
Advances in neural information processing systems, 27, 2014.

Goodfellow, I., Bengio, Y., and Courville, A.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

Gopnik, A.
The philosophical baby.
St Martin's Press, New York, NY, July 2010.

Gopnik, A.
Childhood as a solution to explore–exploit tensions.
Philosophical Transactions of the Royal Society B: Biological Sciences, 375(1803):20190502, June 2020.
doi: 10.1098/rstb.2019.0502.
URL https://doi.org/10.1098/rstb.2019.0502.

Gori, M., Monfardini, G., and Scarselli, F.
A new model for learning in graph domains.
In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734 vol. 2, July 2005.
doi: 10.1109/IJCNN.2005.1555942.

Gottlieb, J. and Oudeyer, P.-Y.
Towards a neuroscience of active sampling and curiosity.
Nature Reviews Neuroscience, 19(12):758–770, nov 2018.
doi: 10.1038/s41583-018-0078-0.
URL https://doi.org/10.1038%2Fs41583-018-0078-0.

Gould, S. J. and Eldredge, N.
Punctuated equilibria: the tempo and mode of evolution reconsidered.
Paleobiology, 3(2):115–151, 1977.
doi: 10.1017/S0094837300005224.

Green, E. J. and Quilty-Dunn, J.
what is an object file?
The British Journal for the Philosophy of Science, 12 2017.
ISSN 0007-0882.
doi: 10.1093/bjps/axx055.
URL https://doi.org/10.1093/bjps/axx055.
axx055.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A.
Multi-object representation learning with iterative variational inference, 2020.

Grisoni, F., Moret, M., Lingwood, R., and Schneider, G.
Bidirectional molecule generation with recurrent neural networks.
Journal of Chemical Information and Modeling, 60(3):1175–1183, January 2020.
doi: 10.1021/acs.jcim.9b00943.
URL https://doi.org/10.1021/acs.jcim.9b00943.

Grizou, J., Points, L. J., Sharma, A., and Cronin, L.
A curious formulation robot enables the discovery of a novel protocell behavior.
Science Advances, 6(5):eaay4237, 2020.
doi: 10.1126/sciadv.aay4237.
URL https://www.science.org/doi/abs/10.1126/sciadv.aay4237.

Hadsell, R., Chopra, S., and LeCun, Y.
Dimensionality reduction by learning an invariant mapping.

In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742, 2006.
doi: 10.1109/CVPR.2006.100.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T.
Mastering diverse domains through world models.
arXiv preprint arXiv:2301.04104, 2023.

Haluptzok, P., Bowers, M., and Kalai, A. T.
Language models can teach themselves to program better.
arXiv preprint arXiv:2207.14502, 2022.

Haluptzok, P., Bowers, M., and Kalai, A. T.
Language Models Can Teach Themselves to Program Better, April 2023.
URL http://arxiv.org/abs/2207.14502.
arXiv:2207.14502 [cs].

Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W.
Relational inductive bias for physical construction in humans and machines, 2018.

Harnad, S.
The symbol grounding problem.
Physica D: Nonlinear Phenomena, 42(1):335–346, 1990.
ISSN 0167-2789.
doi: https://doi.org/10.1016/0167-2789(90)90087-6.
URL https://www.sciencedirect.com/science/article/pii/0167278990900876.

Hauser, M. D., Chomsky, N., and Fitch, W. T.
The faculty of language: What is it, who has it, and how did it evolve?
Science, 298(5598):1569–1579, 2002.
doi: 10.1126/science.298.5598.1569.
URL https://www.science.org/doi/abs/10.1126/science.298.5598.1569.

Hausknecht, M. and Stone, P.
Deep recurrent q-learning for partially observable mdps.
In *2015 aaai fall symposium series*, 2015.

Hausknecht, M., Ammanabrolu, P., Côté, M.-A., and Yuan, X.
Interactive fiction games: A colossal adventure.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7903–7910, 2020.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M.
Deep reinforcement learning with a natural language action space.
In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1621–1630, Berlin, Germany, August 2016a. Association for Computational Linguistics.
doi: 10.18653/v1/P16-1153.
URL https://aclanthology.org/P16-1153.

He, K., Zhang, X., Ren, S., and Sun, J.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016b.

Henrich, J.
The weirdest people in the world.
Farrar, Straus and Giroux, September 2023.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., and Blunsom, P.
Grounded language learning in a simulated 3d world, 2017.

Herrmann, V., Kirsch, L., and Schmidhuber, J.
Learning one abstract bit at a time through self-invented experiments encoded as neural networks.
arXiv preprint arXiv:2212.14374, 2022.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D.
Rainbow: Combining improvements in deep reinforcement learning.
In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S.
Gans trained by a two time-scale update rule converge to a local nash equilibrium.
Advances in neural information processing systems, 30, 2017.

Hill, F., Lampinen, A., Schneider, R., Clark, S., Botvinick, M., McClelland, J. L., and Santoro, A.
Environmental drivers of systematicity and generalization in a situated agent, 2020.

Hill, M. O.
Diversity and evenness: a unifying notation and its consequences.
Ecology, 54(2):427–432, 1973.

Hinaut, X., Petit, M., Pointeau, G., and Dominey, P. F.
Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks.
Frontiers in neurorobotics, 8:16, 2014.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R.
Improving neural networks by preventing co-adaptation of feature detectors.
arXiv preprint arXiv:1207.0580, 2012.

Ho, J., Jain, A., and Abbeel, P.
Denoising diffusion probabilistic models.
Advances in neural information processing systems, 33:6840–6851, 2020.

Hochreiter, S. and Schmidhuber, J.
Long short-term memory.
Neural computation, 9:1735–80, 12 1997.
doi: 10.1162/neco.1997.9.8.1735.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al.
Training compute-optimal large language models.
arXiv preprint arXiv:2203.15556, 2022.

Hornik, K., Stinchcombe, M., and White, H.
Multilayer feedforward networks are universal approximators.
Neural Networks, 2(5):359–366, 1989.
ISSN 0893-6080.
doi: https://doi.org/10.1016/0893-6080(89)90020-8.
URL https://www.sciencedirect.com/science/article/pii/0893608089900208.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I.
Language models as zero-shot planners: Extracting actionable knowledge for embodied agents.
arXiv preprint arXiv:2201.07207, 2022a.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L., Levine, S., Hausman, K., and brian ichter.
Inner monologue: Embodied reasoning through planning with language models.
In *6th Annual Conference on Robot Learning*, 2022b.
URL https://openreview.net/forum?id=3R3Pz5i0tye.

Hudson, D. A. and Manning, C. D.
Learning by abstraction: The neural state machine.
In *Neural Information Processing Systems*, 2019.

Hui, D. Y.-T., Chevalier-Boisvert, M., Bahdanau, D., and Bengio, Y.
Babyai 1.1, 2020.

Hull, C. L.
Principles of behavior: an introduction to behavior theory.
Appleton-Century, 1943.

Hupkes, D., Dankers, V., Mul, M., and Bruni, E.
Compositionality decomposed: how do neural networks generalise?, 2020.

Ioffe, S. and Szegedy, C.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K.
Reinforcement learning with unsupervised auxiliary tasks.
arXiv preprint arXiv:1611.05397, 2016.

Jain, S. and Wallace, B. C.
Attention is not explanation.
arXiv preprint arXiv:1902.10186, 2019.

Janner, M., Li, Q., and Levine, S.
Reinforcement learning as one big sequence modeling problem.
In *Neural Information Processing Systems*, 2021.

Jansen, P.
A systematic survey of text worlds as embodied natural language environments.
In Cote, M.-A., Yuan, X., and Ammanabrolu, P. (eds.), *Wordplay 2022 - 3rd Wordplay*, Wordplay 2022 - 3rd Wordplay: When Language Meets Games Workshop, Proceedings of the Workshop, pp. 1–15. Association for Computational Linguistics (ACL), 2022.
Publisher Copyright: © 2022 Association for Computational Linguistics.; 3rd Wordplay: When Language Meets Games Workshop, Wordplay 2022 ; Conference date: 14-07-2022.

Jiang, M., Grefenstette, E., and Rocktäschel, T.
Prioritized level replay.
In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

Jiang, Y., Gu, S., Murphy, K., and Finn, C.
Language as an abstraction for hierarchical deep reinforcement learning, 2019.

Jiang, Y., Gupta, A., Zhang, Z., Wang, G., Dou, Y., Chen, Y., Fei-Fei, L., Anandkumar, A., Zhu, Y., and Fan, L.
Vima: General robot manipulation with multimodal prompts.
arXiv preprint arXiv:2210.03094, 2022.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R.
Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, 2016.

Jost, L.
Entropy and diversity.
Oikos, 113(2):363–375, 2006.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R.
Planning and acting in partially observable stochastic domains.
Artificial Intelligence, 101(1-2):99–134, May 1998.
doi: 10.1016/s0004-3702(98)00023-x.
URL https://doi.org/10.1016/s0004-3702(98)00023-x.

Kahneman, D.
Thinking, Fast and Slow.
Farrar, Straus & Giroux, New York, NY, October 2011.

Kahneman, D., Treisman, A., and Gibbs, B. J.
The reviewing of object files: Object-specific integration of information.
Cognitive Psychology, 24(2):175–219, 1992.
ISSN 0010-0285.
doi: https://doi.org/10.1016/0010-0285(92)90007-O.
URL https://www.sciencedirect.com/science/article/pii/0010028592900070.

Kaminski, J., Call, J., and Fischer, J.
Word learning in a domestic dog: Evidence for "fast mapping".
Science, 304(5677):1682–1683, June 2004.
doi: 10.1126/science.1097859.
URL https://doi.org/10.1126/science.1097859.

Kaplan, F. and Oudeyer, P.-Y.
The progress drive hypothesis: an interpretation of early imitation, pp. 361–378.
Cambridge University Press, 2007.
doi: 10.1017/CBO9780511489808.024.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D.
Scaling laws for neural language models.
arXiv preprint arXiv:2001.08361, 2020.

Katz, J. and Wright, A.
Same/different abstract-concept learning by pigeons.
Journal of experimental psychology. Animal behavior processes, 32:80–6, 02 2006.
doi: 10.1037/0097-7403.32.1.80.

Keriven, N. and Peyré, G.
Universal invariant and equivariant graph neural networks.
CoRR, abs/1905.04943, 2019.
URL http://arxiv.org/abs/1905.04943.

Kidd, C. and Hayden, B. Y.
The psychology and neuroscience of curiosity.
Neuron, 88(3):449–460, November 2015a.
doi: 10.1016/j.neuron.2015.09.010.
URL https://doi.org/10.1016/j.neuron.2015.09.010.

Kidd, C. and Hayden, B. Y.
The psychology and neuroscience of curiosity.
Neuron, 88(3):449–460, November 2015b.
ISSN 0896-6273.
doi: 10.1016/j.neuron.2015.09.010.
URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4635443/.

Kidger, P. and Lyons, T.
Universal approximation with deep narrow networks.
In *Conference on learning theory*, pp. 2306–2327. PMLR, 2020.

Kim, J., Ricci, M., and Serre, T.
Not-so-clevr: learning same–different relations strains feedforward neural networks.
Interface Focus, 8(4):20180011, 2018.
doi: 10.1098/rsfs.2018.0011.
URL https://royalsocietypublishing.org/doi/abs/10.1098/rsfs.2018.0011.

Kim, T., Choi, J., Edmiston, D., and Lee, S.-g.
Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction.
arXiv preprint arXiv:2002.00737, 2020.

Kingma, D. P. and Ba, J.
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980, 2014.

Kingma, D. P. and Ba, J.
Adam: A method for stochastic optimization, 2017.

Kipf, T. N. and Welling, M.
Semi-supervised classification with graph convolutional networks.
CoRR, abs/1609.02907, 2016.
URL http://arxiv.org/abs/1609.02907.

Kipf, T. N., van der Pol, E., and Welling, M.
Contrastive learning of structured world models.
CoRR, abs/1911.12247, 2019.
URL http://arxiv.org/abs/1911.12247.

Kirby, S.
Transitions: The Evolution of Linguistic Replicators, pp. 121–138.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
ISBN 978-3-642-36086-2.
doi: 10.1007/978-3-642-36086-2_6.
URL https://doi.org/10.1007/978-3-642-36086-2_6.

Kirby, S., Griffiths, T., and Smith, K.
Iterated learning and the evolution of language.
Current Opinion in Neurobiology, 28:108–114, October 2014.
doi: 10.1016/j.conb.2014.07.014.
URL https://doi.org/10.1016/j.conb.2014.07.014.

Koeslag, J. H.
Koinophilia groups sexual creatures into species, promotes stasis, and stabilizes social behaviour.
Journal of Theoretical Biology, 144(1):15–35, May 1990.
doi: 10.1016/s0022-5193(05)80297-8.
URL https://doi.org/10.1016/s0022-5193(05)80297-8.

Koeslag, J. H.
On the engine of speciation.
Journal of Theoretical Biology, 177(4):401–409, December 1995.
doi: 10.1006/jtbi.1995.0256.
URL https://doi.org/10.1006/jtbi.1995.0256.

Kojima, T.
Generalization between productive use and receptive discrimination of names in an artificial visual language by a chimpanzee.

International Journal of Primatology, 5(2):161–182, April 1984.
doi: 10.1007/bf02735739.
URL https://doi.org/10.1007/bf02735739.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y.
Large language models are zero-shot reasoners.
In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.
URL https://openreview.net/forum?id=e2TBb5y0yFf.

Kovač, G., Sawayama, M., Portelas, R., Colas, C., Dominey, P. F., and Oudeyer, P.-Y.
Large language models as superpositions of cultural perspectives.
arXiv preprint arXiv:2307.07870, 2023.

Krizhevsky, A., Sutskever, I., and Hinton, G. E.
Imagenet classification with deep convolutional neural networks.
In Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Lake, B. M. and Baroni, M.
Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks, 2018.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J.
Building machines that learn and think like people.
Behavioral and Brain Sciences, 40, Nov 2016.
ISSN 1469-1825.
doi: 10.1017/s0140525x16001837.
URL http://dx.doi.org/10.1017/S0140525X16001837.

Laland, K. N. and O'Brien, M. J.
Cultural niche construction: An introduction.
Biological Theory, 6(3):191–202, September 2011.
doi: 10.1007/s13752-012-0026-6.
URL https://doi.org/10.1007/s13752-012-0026-6.

Lampinen, A. K., Roy, N., Dasgupta, I., Chan, S. C., Tam, A., Mcclelland, J., Yan, C., Santoro, A., Rabinowitz, N. C., Wang, J., et al.
Tell me why! explanations support learning relational and causal structure.
In *International Conference on Machine Learning*, pp. 11868–11890. PMLR, 2022.

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X.
Hypertree proof search for neural theorem proving.
Advances in Neural Information Processing Systems, 35:26337–26349, 2022.

Laversanne-Finot, A., Pere, A., and Oudeyer, P.-Y.
Curiosity driven exploration of learned disentangled goal spaces.
In *Conference on Robot Learning*, pp. 487–504. PMLR, 2018.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. H.
Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022.
URL https://arxiv.org/abs/2207.01780.

Lecun, Y. and Bengio, Y.
Convolutional networks for images, speech, and time-series.
01 1995.

LeCun, Y., Bengio, Y., and Hinton, G.
Deep learning.
Nature, 521(7553):436–444, May 2015.
doi: 10.1038/nature14539.
URL https://doi.org/10.1038/nature14539.

Legg, S., Hutter, M., et al.
A collection of definitions of intelligence.
Frontiers in Artificial Intelligence and applications, 157:17, 2007.

Lehman, J. and Stanley, K.
Novelty Search and the Problem with Objectives, pp. 37–56.
11 2011a.
ISBN 978-1-4614-1769-9.
doi: 10.1007/978-1-4614-1770-5_3.

Lehman, J. and Stanley, K. O.
Evolving a diversity of virtual creatures through novelty search and local competition.
In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pp. 211–218, New York, NY, USA, 2011b. Association for Computing Machinery.
ISBN 9781450305570.
doi: 10.1145/2001576.2001606.
URL https://doi.org/10.1145/2001576.2001606.

Lehman, J. and Stanley, K. O.
Abandoning objectives: evolution through the search for novelty alone.
Evol. Comput., 19(2):189–223, February 2011c.

Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., and Stanley, K. O.
Evolution through Large Models, June 2022.
URL http://arxiv.org/abs/2206.08896.
arXiv:2206.08896 [cs].

Lenat, D. B.
Am, an artificial intelligence approach to discovery in mathematics as heuristic search.
1976.
URL https://api.semanticscholar.org/CorpusID:60603795.

Lenat, D. B.
Eurisko: A program that learns new heuristics and domain concepts.
Artificial Intelligence, 21(1-2):61–98, March 1983.

doi: 10.1016/s0004-3702(83)80005-8.
URL https://doi.org/10.1016/s0004-3702(83)80005-8.

Lenat, D. B.
CYC.
Communications of the ACM, 38(11):33–38, November 1995.
doi: 10.1145/219717.219745.
URL https://doi.org/10.1145/219717.219745.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P.
Graph matching networks for learning the similarity of graph structured objects.
CoRR, abs/1904.12787, 2019.
URL http://arxiv.org/abs/1904.12787.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A.
Code as policies: Language model programs for embodied control.
arXiv preprint arXiv:2209.07753, 2022a.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al.
Holistic evaluation of language models.
arXiv preprint arXiv:2211.09110, 2022b.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K.
Let's verify step by step.
arXiv preprint arXiv:2305.20050, 2023.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D.
Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971, 2015.

Linke, C., Ady, N. M., White, M., Degris, T., and White, A.
Adapting behavior via intrinsic reward: A survey and empirical study.
Journal of artificial intelligence research, 69:1287–1332, 2020.

Littman, M. L.
Markov games as a framework for multi-agent reinforcement learning.
In *International Conference on Machine Learning*, 1994.
URL https://api.semanticscholar.org/CorpusID:8108362.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T.
Object-centric learning with slot attention.
CoRR, abs/2006.15055, 2020.
URL https://arxiv.org/abs/2006.15055.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T.
A survey of reinforcement learning informed by natural language, 2019.

Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., and Jiang, D.
Wizardcoder: Empowering code large language models with evol-instruct.
arXiv preprint arXiv:2306.08568, 2023.

Lupyan, G., Rahman, R. A., Boroditsky, L., and Clark, A.
Effects of language on visual perception.
Trends in Cognitive Sciences, 24(11):930–944, November 2020.
doi: 10.1016/j.tics.2020.08.005.
URL https://doi.org/10.1016/j.tics.2020.08.005.

Lynch, C. and Sermanet, P.
Language conditioned imitation learning over unstructured data.
arXiv preprint arXiv:2005.07648, 2020.

Ma, G., Ahmed, N. K., Willke, T. L., and Yu, P. S.
Deep graph similarity learning: A survey.
CoRR, abs/1912.11615, 2019.
URL http://arxiv.org/abs/1912.11615.

MacQueen, J.
Some methods for classification and analysis of multivariate observations.
1967.
Published: Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967).

Madaan, A., Zhou, S., Alon, U., Yang, Y., and Neubig, G.
Language models of code are few-shot commonsense learners.
In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 1384–1403, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
doi: 10.18653/v1/2022.emnlp-main.90.
URL https://aclanthology.org/2022.emnlp-main.90.

Madotto, A., Namazifar, M., Huizinga, J., Molino, P., Ecoffet, A., Zheng, H., Yu, D., Papangelis, A., Khatri, C., and Tur, G.
Exploration based language learning for text-based games.
In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021.
ISBN 9780999241165.

Mahr, J. and Schacter, D. L.
A language of episodic thought? (commentary on quilty-dunn et al.).
May 2023.
doi: 10.31234/osf.io/8p3cb.
URL https://doi.org/10.31234/osf.io/8p3cb.

Mangin, O., Filliat, D., Ten Bosch, L., and Oudeyer, P.-Y.
Mca-nmf: Multimodal concept acquisition with non-negative matrix factorization.
PloS one, 10(10):e0140732, 2015.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., Köppe, T., Millikin, K., Gaffney, S., Elster, S., Broshear, J., Gamble, C., Milan, K., Tung, R., Hwang, M., Cemgil, T., Barekatain, M., Li, Y., Mandhane, A., Hubert, T., Schrittwieser, J., Hassabis, D., Kohli, P., Riedmiller, M., Vinyals, O., and Silver, D.
Faster sorting algorithms discovered using deep reinforcement learning.
Nature, 618(7964):257–263, June 2023.
doi: 10.1038/s41586-023-06004-9.
URL https://doi.org/10.1038/s41586-023-06004-9.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y.
On the universality of invariant networks.
CoRR, abs/1901.09342, 2019.
URL http://arxiv.org/abs/1901.09342.

Marzoev, A., Madden, S., Kaashoek, M. F., Cafarella, M., and Andreas, J.
Unnatural language processing: Bridging the gap between synthetic and natural language data, 2020.

Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A.
Monet: Unsupervised scene decomposition and representation.
CoRR, abs/1901.11390, 2019.
URL http://arxiv.org/abs/1901.11390.

McCulloch, W. S. and Pitts, W.
A logical calculus of the ideas immanent in nervous activity.
The Bulletin of Mathematical Biophysics, 5(4):115–133, December 1943.
doi: 10.1007/bf02478259.
URL https://doi.org/10.1007/bf02478259.

Medam, T., Marzouki, Y., Montant, M., and Fagot, J.
Categorization does not promote symmetry in guinea baboons (papio papio).
Animal Cognition, 19(5):987–998, June 2016.
doi: 10.1007/s10071-016-1003-4.
URL https://doi.org/10.1007/s10071-016-1003-4.

Meyer, J.-A. and Wilson, S. W.
A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers, pp. 222–227.
1991.

Meyerson, E., Nelson, M. J., Bradley, H., Moradi, A., Hoover, A. K., and Lehman, J.
Language model crossover: Variation through few-shot prompting.
arXiv preprint arXiv:2302.12170, 2023.

Michelmann, S., Kumar, M., Norman, K. A., and Toneva, M.
Large language models can segment narrative events similarly to humans.
arXiv preprint arXiv:2301.10297, 2023.

Miller, G. A.

The magical number seven, plus or minus two: Some limits on our capacity for processing information.
Psychological Review, 63(2):81–97, March 1956.
doi: 10.1037/h0043158.
URL https://doi.org/10.1037/h0043158.

Minsky, M. and Papert, S.
Perceptrons.
MIT Press, London, England, February 1969.

Mirchandani, S., Karamcheti, S., and Sadigh, D.
Ella: Exploration through learned language abstraction.
Advances in Neural Information Processing Systems, 34:29529–29540, 2021.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Bae, S., et al.
Chip placement with deep reinforcement learning.
arXiv preprint arXiv:2004.10746, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al.
Human-level control through deep reinforcement learning.
nature, 518(7540):529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K.
Asynchronous methods for deep reinforcement learning.
In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Molinaro, G. and Collins, A. G.
A goal-centric outlook on learning.
Trends in Cognitive Sciences, sep 2023.
doi: 10.1016/j.tics.2023.08.011.
URL https://doi.org/10.1016%2Fj.tics.2023.08.011.

Moravec, H.
The role of raw power in intelligence.
Unpublished maniscript, available at https://web.archive.org/web/20160303232511/http://www.frc.ri.cmu.edu/users/hpm/project.archive/general.articles/1975/Raw.Power.html (Accessed 2023/10/02), 1976.

Morimoto, J. and Doya, K.
Reinforcement learning of dynamic motor sequence: learning to stand up.
In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, volume 3, pp. 1721–1726 vol.3, 1998.
doi: 10.1109/IROS.1998.724846.

Moura, L. and Ullrich, S.
The Lean 4 Theorem Prover and Programming Language, pp. 625–635.
07 2021.
ISBN 978-3-030-79875-8.
doi: 10.1007/978-3-030-79876-5_37.

Mouret, J.-B. and Clune, J.
Illuminating search spaces by mapping elites.
arXiv preprint arXiv:1504.04909, 2015a.

Mouret, J.-B. and Clune, J.
Illuminating search spaces by mapping elites.
arXiv preprint arXiv:1504.04909, 2015b.

Mu, J., Zhong, V., Raileanu, R., Jiang, M., Goodman, N., Rocktäschel, T., and Grefenstette, E.
Improving intrinsic exploration with language abstractions.
Advances in Neural Information Processing Systems, 35:33947–33960, 2022.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S.
Visual reinforcement learning with imagined goals.
Advances in neural information processing systems, 31, 2018.

Narasimhan, K., Kulkarni, T., and Barzilay, R.
Language understanding for text-based games using deep reinforcement learning.
In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11, Lisbon, Portugal, September 2015. Association for Computational Linguistics.
doi: 10.18653/v1/D15-1001.
URL https://aclanthology.org/D15-1001.

Newell, A., Shaw, J. C., and Simon, H. A.
Report on a general problem-solving program.
In *IFIP Congress*, 1959.
URL https://api.semanticscholar.org/CorpusID:35199622.

Ng, A. Y. and Russell, S. J.
Algorithms for inverse reinforcement learning.
In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pp. 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
ISBN 1558607072.

Ng, A. Y., Russell, S., et al.
Algorithms for inverse reinforcement learning.
In *Icml*, volume 1, pp. 2, 2000.

Nguyen, K., Misra, D., Schapire, R., Dudík, M., and Shafto, P.
Interactive learning from activity description, 2021.

Nieder, A.
Prefrontal cortex and the evolution of symbolic reference.
Current Opinion in Neurobiology, 19(1):99–108, 2009.

ISSN 0959-4388.
doi: https://doi.org/10.1016/j.conb.2009.04.008.
URL https://www.sciencedirect.com/science/article/pii/S095943880900035X.
Cognitive neuroscience.

Nisioti, E., Mahaut, M., Oudeyer, P.-Y., Momennejad, I., and Moulin-Frier, C.
Social network structure shapes innovation: experience-sharing in rl with sapiens.
arXiv preprint arXiv:2206.05060, 2022.

Norvig, P.
On chomsky and the two cultures of statistical learning.
2011.
URL http://norvig.com/chomsky.html.

Odling-Smee, F. J., Laland, K. N., and Feldman, M. W.
Niche construction.
Monographs in Population Biology. Princeton University Press, Princeton, NJ, July 2003.

Olah, C., Mordvintsev, A., and Schubert, L.
Feature visualization.
Distill, 2(11), November 2017.
doi: 10.23915/distill.00007.
URL https://doi.org/10.23915/distill.00007.

O'Neil, C.
Weapons of math destruction.
Penguin Books, Harlow, England, June 2017.

OpenAI.
Gpt-4 technical report, 2023.

OpenAI, O., Plappert, M., Sampedro, R., Xu, T., Akkaya, I., Kosaraju, V., Welinder, P., D'Sa, R., Petron, A., d. O. Pinto, H. P., Paino, A., Noh, H., Weng, L., Yuan, Q., Chu, C., and Zaremba, W.
Asymmetric self-play for automatic goal discovery in robotic manipulation, 2021.

Oudeyer, P.-Y. and Kaplan, F.
What is intrinsic motivation? a typology of computational approaches.
Frontiers in Neurorobotics, 1, 2007a.
ISSN 1662-5218.
doi: 10.3389/neuro.12.006.2007.
URL https://www.frontiersin.org/articles/10.3389/neuro.12.006.2007.

Oudeyer, P.-Y. and Kaplan, F.
What is intrinsic motivation? a typology of computational approaches.
Frontiers in neurorobotics, 1:6, 2007b.

Oudeyer, P.-Y. and Kaplan, F.
What is intrinsic motivation? a typology of computational approaches.
Frontiers in neurorobotics, pp. 6, 2007c.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R.
Training language models to follow instructions with human feedback.
In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022.
URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al.
Stabilizing transformers for reinforcement learning.
In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Pascanu, R., Mikolov, T., and Bengio, Y.
On the difficulty of training recurrent neural networks.
In Dasgupta, S. and McAllester, D. (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.
URL https://proceedings.mlr.press/v28/pascanu13.html.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.
Pytorch: An imperative style, high-performance deep learning library.
Advances in neural information processing systems, 32, 2019.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T.
Curiosity-driven exploration by self-supervised prediction.
In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2778–2787. JMLR.org, 2017.

Pathak, D., Gandhi, D., and Gupta, A.
Self-supervised exploration via disagreement.
In *ICML*, 2019.

Patil, G. and Taillie, C.
Diversity as a concept and its measurement.
Journal of the American statistical Association, 77(379):548–561, 1982.

Paul, R., Arkin, J., Roy, N., and Howard, T.
Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators.
In *Robotics: Science and Systems*, 2016.

Pepperberg, I. M.
The Alex Studies.
Harvard University Press, December 2000.
doi: 10.4159/9780674041998.
URL https://doi.org/10.4159/9780674041998.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A.
Film: Visual reasoning with a general conditioning layer, 2017.

Perry, S., Carter, A., Smolla, M., Akçay, E., Nöbel, S., Foster, J. G., and Healy, S. D.
Not by transmission alone: the role of invention in cultural evolution.
Philosophical Transactions of the Royal Society B: Biological Sciences, 376(1828), May 2021.
doi: 10.1098/rstb.2020.0049.
URL https://doi.org/10.1098/rstb.2020.0049.

Pong, V., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S.
Skew-fit: State-covering self-supervised reinforcement learning.
In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7783–7792. PMLR, 13–18 Jul 2020.
URL https://proceedings.mlr.press/v119/pong20a.html.

Portelas, R., Colas, C., Hofmann, K., and Oudeyer, P.-Y.
Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments.
In Kaelbling, L. P., Kragic, D., and Sugiura, K. (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 835–853. PMLR, 30 Oct–01 Nov 2020.
URL https://proceedings.mlr.press/v100/portelas20a.html.

Portelas, R., Colas, C., Weng, L., Hofmann, K., and Oudeyer, P.-Y.
Automatic curriculum learning for deep rl: A short survey.
In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021.
ISBN 9780999241165.

Premack, A. J. and Premack, D.
Teaching language to an ape.
Scientific American, 227(4):92–99, October 1972.
doi: 10.1038/scientificamerican1072-92.
URL https://doi.org/10.1038/scientificamerican1072-92.

Puebla, G. and Bowers, J. S.
Can deep convolutional neural networks learn same-different relations?
bioRxiv, 2021.
doi: 10.1101/2021.04.06.438551.
URL https://www.biorxiv.org/content/early/2021/05/12/2021.04.06.438551.

Pugh, J., Soros, L., and Stanley, K.
Quality Diversity: A New Frontier for Evolutionary Computation.
Frontiers in Robotics and AI, 3, July 2016.
doi: 10.3389/frobt.2016.00040.

Pylyshyn, Z. W.
Things and places: How the mind connects with the world.
MIT press, 2007.

Péré, A., Forestier, S., Sigaud, O., and Oudeyer, P.-Y.
Unsupervised learning of goal spaces for intrinsically motivated goal exploration.
In *International Conference on Learning Representations*, 2018.
URL https://openreview.net/forum?id=S1DWPP1A-.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J.
Pointnet: Deep learning on point sets for 3d classification and segmentation.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Quilty-Dunn, J., Porot, N., and Mandelbaum, E.
The best game in town: The re-emergence of the language of thought hypothesis across the cognitive sciences.
Behavioral and Brain Sciences, pp. 1–55, December 2022.
doi: 10.1017/s0140525x22002849.
URL https://doi.org/10.1017/s0140525x22002849.

Radford, A. and Narasimhan, K.
Improving language understanding by generative pre-training.
2018.

Radford, A., Jozefowicz, R., and Sutskever, I.
Learning to generate reviews and discovering sentiment.
arXiv preprint arXiv:1704.01444, 2017.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I.
Language models are unsupervised multitask learners.
2019.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.
Learning transferable visual models from natural language supervision.
In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al.
Scaling language models: Methods, analysis & insights from training gopher.
arXiv preprint arXiv:2112.11446, 2021.

Ramamurthy, R., Ammanabrolu, P., Brantley, K., Hessel, J., Sifa, R., Bauckhage, C., Hajishirzi, H., and Choi, Y.
Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization.
2022.
URL https://arxiv.org/abs/2210.01241.

Reimers, N. and Gurevych, I.
Sentence-BERT: Sentence embeddings using Siamese BERT-networks.
In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

doi: 10.18653/v1/D19-1410.
URL https://aclanthology.org/D19-1410.

Reinke, C., Etcheverry, M., and Oudeyer, P.-Y.
Intrinsically motivated discovery of diverse patterns in self-organizing systems.
arXiv preprint arXiv:1908.06663, 2019.

Reinke, C., Etcheverry, M., and Oudeyer, P.-Y.
Intrinsically Motivated Discovery of Diverse Patterns in Self-Organizing Systems.
In *International Conference on Learning Representations (ICLR)*, 2020.

Rescorla, M.
The Language of Thought Hypothesis.
In Zalta, E. N. (ed.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition, 2019.

Ricci, M., Cadène, R., and Serre, T.
Same-different conceptualization: a machine vision perspective.
Current Opinion in Behavioral Sciences, 37:47–55, 2021.
ISSN 2352-1546.
doi: https://doi.org/10.1016/j.cobeha.2020.08.008.
URL https://www.sciencedirect.com/science/article/pii/S2352154620301352.
Same-different conceptualization.

Richards, D. G., Wolz, J. P., and Herman, L. M.
Vocal mimicry of computer-generated sounds and vocal labeling of objects by a bottlenosed dolphin, tursiops truncatus..
Journal of Comparative Psychology, 98(1):10–28, 1984.
doi: 10.1037/0735-7036.98.1.10.
URL https://doi.org/10.1037/0735-7036.98.1.10.

Rochat, P. and Striano, T.
Perceived self in infancy.
Infant Behavior and Development, 23(3-4):513–530, mar 2000.
doi: 10.1016/s0163-6383(01)00055-8.
URL https://doi.org/10.1016%2Fs0163-6383%2801%2900055-8.

Rogers, A., Kovaleva, O., and Rumshisky, A.
A primer in bertology: What we know about how bert works.
Transactions of the Association for Computational Linguistics, 8:842–866, 2021.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., and Wayne, G.
Experience replay for continual learning.
Advances in Neural Information Processing Systems, 32, 2019.

Rosa, R. and Mareček, D.
Inducing syntactic trees from bert representations.
arXiv preprint arXiv:1906.11511, 2019.

Rosenblatt, F.
The perceptron - a perceiving and recognizing automaton.
Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D., and Lake, B. M.
A benchmark for systematic generalization in grounded language understanding, 2020.

Rule, J., Goddu, M. K., Chu, J., Pinter, V., Reagan, E. R., Bonawitz, E., alison gopnik, and Ullman, T. D.
Fun isn't easy: Children choose more difficult options when "playing for fun" vs. "trying to win".
jul 2023.
doi: 10.31234/osf.io/q7wh4.
URL https://doi.org/10.31234%2Fosf.io%2Fq7wh4.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J.
Learning representations by back-propagating errors.
Nature, 323(6088):533–536, October 1986a.
doi: 10.1038/323533a0.
URL https://doi.org/10.1038/323533a0.

Rumelhart, D. E., McClelland, J. L., and The PDP Research Group.
Parallel distributed processing.
Parallel Distributed Processing. MIT Press, London, England, January 1986b.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.
Imagenet large scale visual recognition challenge.
International journal of computer vision, 115:211–252, 2015.

Russell, S.
Human compatible.
Viking, October 2019.

Russell, S. and Norvig, P.
Artificial intelligence.
Pearson, Upper Saddle River, NJ, 4 edition, November 2020.

Salge, C., Glackin, C., and Polani, D.
Empowerment – an introduction.
10 2013.
doi: 10.1007/978-3-642-53734-9_4.

Samuel, A. L.
Some studies in machine learning using the game of checkers.
IBM Journal of Research and Development, 3(3):210–229, July 1959.
doi: 10.1147/rd.33.0210.
URL https://doi.org/10.1147/rd.33.0210.

Sancaktar, C., Piater, J., and Martius, G.
Regularity as intrinsic reward for free play.
2023.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T.
A simple neural network module for relational reasoning.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4967–4976. Curran Associates, Inc., 2017a.
URL http://papers.nips.cc/paper/7082-a-simple-neural-network-module-for-relational-reasoning.pdf.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T.
A simple neural network module for relational reasoning, 2017b.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P. W., and Lillicrap, T. P.
A simple neural network module for relational reasoning.
In *NIPS*, 2017c.

Savage-Rumbaugh, S., McDonald, K., Sevcik, R. A., Hopkins, W. D., and Rubert, E.
Spontaneous symbol acquisition and communicative use by pygmy chimpanzees (pan paniscus).
Journal of Experimental Psychology: General, 115(3):211–235, 1986.
doi: 10.1037/0096-3445.115.3.211.
URL https://doi.org/10.1037/0096-3445.115.3.211.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G.
The graph neural network model.
IEEE Transactions on Neural Networks, 20(1):61–80, Jan 2009.
ISSN 1941-0093.
doi: 10.1109/TNN.2008.2005605.

Schacter, D. L.
Constructive memory: past and future.
Dialogues in Clinical Neuroscience, 14(1):7–18, March 2012.
doi: 10.31887/dcns.2012.14.1/dschacter.
URL https://doi.org/10.31887/dcns.2012.14.1/dschacter.

Schaul, T., Horgan, D., Gregor, K., and Silver, D.
Universal value function approximators.
In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015a. PMLR.
URL https://proceedings.mlr.press/v37/schaul15.html.

Schaul, T., Horgan, D., Gregor, K., and Silver, D.
Universal value function approximators.
In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015b.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D.
Prioritized experience replay.
arXiv preprint arXiv:1511.05952, 2015c.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T.
Toolformer: Language models can teach themselves to use tools, 2023.
URL https://arxiv.org/abs/2302.04761.

Schmidhuber, J.
Artificial curiosity creativity since 1990-91.
Available at https://people.idsia.ch/~juergen/artificial-curiosity-since-1990.html (Accessed 2023/10/02).

Schmidhuber, J.
Formal theory of creativity, fun, and intrinsic motivation (1990–2010).
IEEE Transactions on Autonomous Mental Development, 2(3):230–247, sep 2010.
doi: 10.1109/tamd.2010.2056368.
URL https://doi.org/10.1109%2Ftamd.2010.2056368.

Schmidhuber, J.
Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem, 2011.
URL https://arxiv.org/abs/1112.5309.

Schmidhuber, J.
Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem.
Frontiers in psychology, 4:313, 2013.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P.
Trust region policy optimization.
In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O.
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347, 2017.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P.
High-dimensional continuous control using generalized advantage estimation, 2018.

Schulman, J., Zoph, B., Kim, C., Hilton, J., Menick, J., Weng, J., Uribe, J. F. C., Fedus, L., Metz, L., Pokorny, M., Lopes, R. G., Zhao, S., Vijayvergiya, A., Sigler, E., Perelman, A., Voss, C., Heaton, M., Parish, J., Cummings, D., Nayak, R., Balcom, V., Schnurr, D., Kaftan, T., Hallacy, C., Turley, N., Deutsch, N., Goel, V., Ward, J., Konstantinidis, A., Zaremba, W., Ouyang, L., Bogdonoff, L., Gross, J., Medina, D., Yoo, S., Lee, T., Lowe, R., Mossing, D., Huizinga, J., Jiang, R., Wainwright, C., Almeida, D., Lin, S., Zhang, M., Xiao, K., Slama, K., Bills, S., Gray, A., Leike, J., Pachocki, J., Tillet, P., Jain, S., Brockman, G., Ryder, N., Paino, A., Yuan, Q., Winter, C., Wang, B., Bavarian, M., Babuschkin, I., Sidor, S., Kanitscheider, I., Pavlov, M., Plappert, M., Tezak, N., Jun, H., Zhuk, W., Pong, V., Kaiser, L., Tworek, J., Carr, A., Weng, L., Agarwal, S., Cobbe, K., Kosaraju, V., Power, A., Polu, S., Han, J., Puri, R., Jain, S., Chess, B., Gibson, C., Boiko, O., Parparita, E., Tootoonchian, A., Kosic, K., and Hesse, C.
Introducing chatgpt.
https://openai.com/blog/chatgpt, 2022.
Accessed: 2023-09-25.

Schuster, T., Kalyan, A., Polozov, A., and Kalai, A. T.
Programming Puzzles.
June 2021.
URL https://openreview.net/forum?id=fe_hCc4RBrg.

Schuurmans, D.
Memory augmented large language models are computationally universal.
arXiv preprint arXiv:2301.04589, 2023.

Shannon, C. E.
A mathematical theory of communication.
Bell System Technical Journal, 27(3):379–423, July 1948.
doi: 10.1002/j.1538-7305.1948.tb01338.x.
URL https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Shepard, R. N. and Metzler, J.
Mental rotation of three-dimensional objects.
Science, 171(3972):701–703, 1971.
ISSN 0036-8075.
doi: 10.1126/science.171.3972.701.
URL https://science.sciencemag.org/content/171/3972/701.

Shinn, N., Labash, B., and Gopinath, A.
Reflexion: an autonomous agent with dynamic memory and self-reflection.
arXiv preprint arXiv:2303.11366, 2023.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M.
Alfworld: Aligning text and embodied environments for interactive learning.
arXiv preprint arXiv:2010.03768, 2020.

Sidman, M., Rauzin, R., Lazar, R., Cunningham, S., Tailby, W., and Carrigan, P.
A search for symmetry in the conditional discriminations of rhesus monkeys, baboons, and children.
Journal of the Experimental Analysis of Behavior, 37(1):23–44, January 1982.
doi: 10.1901/jeab.1982.37-23.
URL https://doi.org/10.1901/jeab.1982.37-23.

Sigaud, O., Caselles-Dupré, H., Colas, C., Akakzia, A., Oudeyer, P.-Y., and Chetouani, M.
Towards teachable autonomous agents.
arXiv preprint arXiv:2105.11977, 2021.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D.
Mastering the game of go with deep neural networks and tree search.
Nature, 529(7587):484–489, January 2016.
doi: 10.1038/nature16961.
URL https://doi.org/10.1038/nature16961.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.

Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
arXiv preprint arXiv:1712.01815, 2017.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., and Garg, A.
Progprompt: Generating situated robot task plans using large language models.
arXiv preprint arXiv:2209.11302, 2022.

Singh, S., Barto, A. G., and Chentanez, N.
Intrinsically motivated reinforcement learning.
In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pp. 1281–1288, Cambridge, MA, USA, 2004. MIT Press.

Singh, S., Lewis, R. L., Barto, A. G., and Sorg, J.
Intrinsically motivated reinforcement learning: An evolutionary perspective.
IEEE Transactions on Autonomous Mental Development, 2(2):70–82, 2010.

Smith, J. D., Minda, J. P., and Washburn, D. A.
Category learning in rhesus monkeys: A study of the shepard, hovland, and jenkins (1961) tasks.
Journal of Experimental Psychology: General, 133(3):398–414, 2004.
doi: 10.1037/0096-3445.133.3.398.
URL https://doi.org/10.1037/0096-3445.133.3.398.

Stanley, K. O. and Lehman, J.
Why Greatness Cannot Be Planned.
Springer International Publishing, 2015.
doi: 10.1007/978-3-319-15524-1.
URL https://doi.org/10.1007/978-3-319-15524-1.

Steels, L.
The Autotelic Principle, volume 3139, pp. 629–629.
02 2004.
ISBN 978-3-540-22484-6.
doi: 10.1007/978-3-540-27833-7_17.

Steels, L.
Semiotic dynamics for embodied agents.
IEEE Intelligent Systems, 21(3):32–38, 2006.
doi: 10.1109/MIS.2006.58.

Steels, L. (ed.).
Design Patterns in Fluid Construction Grammar.
Constructional Approaches to Language. John Benjamins Publishing, Amsterdam, Netherlands, December 2011.

Steinhaus, H.
Sur la division des corps matériels en parties.
Bulletin de l'Académie Polonaise des Sciences, Classe 3, 4:801–804, 1957.
ISSN 0001-4095.

Strehl, A. L. and Littman, M. L.
An analysis of model-based interval estimation for markov decision processes.
Journal of Computer and System Sciences, 74(8):1309–1331, 2008.
ISSN 0022-0000.
doi: https://doi.org/10.1016/j.jcss.2007.08.009.
URL https://www.sciencedirect.com/science/article/pii/S0022000008000767.
Learning Theory 2005.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y.
Roformer: Enhanced transformer with rotary position embedding.
arXiv preprint arXiv:2104.09864, 2021.

Sugita, Y. and Tani, J.
Learning semantic combinatoriality from the interaction between linguistic and behavioral processes.
Adaptive behavior, 13(1):33–52, 2005.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R.
Intrinsic motivation and automatic curricula via asymmetric self-play, 2018.

Sutskever, I., Vinyals, O., and Le, Q. V.
Sequence to sequence learning with neural networks.
Advances in neural information processing systems, 27, 2014.

Sutton, R.
The bitter lesson, 2019.
URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Sutton, R. S. and Barto, A. G.
Reinforcement Learning.
Adaptive Computation and Machine Learning series. Bradford Books, Cambridge, MA, 2 edition, November 2018a.

Sutton, R. S. and Barto, A. G.
Reinforcement Learning: An Introduction.
The MIT Press, 2018b.

Sutton, R. S., Barto, A. G., et al.
Introduction to reinforcement learning, volume 135.
MIT press Cambridge, 1998.

Sutton, R. S., Precup, D., and Singh, S.
Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning.
Artificial Intelligence, 112(1):181–211, 1999.
ISSN 0004-3702.
doi: https://doi.org/10.1016/S0004-3702(99)00052-1.
URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Szathmáry, E. and Smith, J. M.
The major evolutionary transitions.
Nature, 374(6519):227–232, March 1995.
doi: 10.1038/374227a0.
URL https://doi.org/10.1038/374227a0.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A.
Going deeper with convolutions.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Tam, A., Rabinowitz, N. C., Lampinen, A. K., Roy, N. A., Chan, S. C., Strouse, D., Wang, J. X., Banino, A., and Hill, F.
Semantic exploration from language abstractions and pretrained representations.
In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.
URL https://openreview.net/forum?id=-NOQJw5z_KY.

Tani, J.
Exploring robotic minds: actions, symbols, and consciousness as self-organizing dynamic phenomena.
Oxford University Press, 2016.

Taniguchi, T., Nagai, T., Nakamura, T., Iwahashi, N., Ogata, T., and Asoh, H.
Symbol emergence in robotics: a survey.
Advanced Robotics, 30(11-12):706–728, 2016.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B.
Stanford alpaca: An instruction-following llama model.
https://github.com/tatsu-lab/stanford_alpaca, 2023.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al.
Deepmind control suite.
arXiv preprint arXiv:1801.00690, 2018.

Team, A. A., Bauer, J., Baumli, K., Baveja, S., Behbahani, F., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., et al.
Human-timescale adaptation in an open-ended task space.
arXiv preprint arXiv:2301.07608, 2023.

Tellex, S., Gopalan, N., Kress-Gazit, H., and Matuszek, C.
Robots that use language.
Annual Review of Control, Robotics, and Autonomous Systems, 3(1):25–55, 2020.
doi: 10.1146/annurev-control-101119-071628.
URL https://doi.org/10.1146/annurev-control-101119-071628.

Tenbrink, T.
Space, Time, and the Use of Language: An Investigation of Relationships.
De Gruyter Mouton, 2008.

ISBN 978-3-11-019882-9.
doi: doi:10.1515/9783110198829.
URL https://doi.org/10.1515/9783110198829.

Tenbrink, T.
Reference frames of space and time in language.
Journal of Pragmatics, 43(3):704–722, 2011.
ISSN 0378-2166.
doi: https://doi.org/10.1016/j.pragma.2010.06.020.
URL https://www.sciencedirect.com/science/article/pii/S037821661000192X.
The Language of Space and Time.

Tenney, I., Das, D., and Pavlick, E.
Bert rediscovers the classical nlp pipeline.
arXiv preprint arXiv:1905.05950, 2019.

Tennie, C., Call, J., and Tomasello, M.
Ratcheting up the ratchet: on the evolution of cumulative culture.
Philosophical Transactions of the Royal Society B: Biological Sciences, 364(1528):2405–2415, 2009.

Terrace, H. S., Petitto, L. A., Sanders, R. J., and Bever, T. G.
Can an ape create a sentence?
Science, 206(4421):891–902, November 1979.
doi: 10.1126/science.504995.
URL https://doi.org/10.1126/science.504995.

Tesauro, G.
Temporal difference learning and td-gammon.
Commun. ACM, 38(3):58–68, mar 1995.
ISSN 0001-0782.
doi: 10.1145/203330.203343.
URL https://doi.org/10.1145/203330.203343.

Thalmann, M., Souza, A. S., and Oberauer, K.
How does chunking help working memory?
Journal of Experimental Psychology: Learning, Memory, and Cognition, 45(1):37–55, January 2019.
doi: 10.1037/xlm0000578.
URL https://doi.org/10.1037/xlm0000578.

Thiel, P. and Masters, B.
Zero to one.
Currency, September 2014.

Todorov, E., Erez, T., and Tassa, Y.
Mujoco: A physics engine for model-based control.
In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.
IEEE, 2012.
doi: 10.1109/IROS.2012.6386109.

Tomasello, M.
Becoming human.
Harvard University Press, London, England, January 2021.

Tomasello, M.
The Evolution of Agency: Behavioral Organization from Lizards to Humans.
MIT Press, 2022.

Tomasello, M., Kruger, A. C., and Ratner, H. H.
Cultural learning.
Behavioral and Brain Sciences, 16(3):495–511, September 1993.
doi: 10.1017/s0140525x0003123x.
URL https://doi.org/10.1017/s0140525x0003123x.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.
Llama: Open and efficient foundation language models.
arXiv preprint arXiv:2302.13971, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T.
Llama 2: Open foundation and fine-tuned chat models, 2023b.

Tuci, E., Ferrauto, T., Zeschel, A., Massera, G., and Nolfi, S.
An experiment on behavior generalization and the emergence of linguistic compositionality in evolving robots.
IEEE Transactions on Autonomous Mental Development, 3(2):176–189, 2011.

Turing, A. M.
On computable numbers, with an application to the entscheidungsproblem.
Proceedings of the London Mathematical Society, s2-42(1):230–265, 1937.
doi: 10.1112/plms/s2-42.1.230.
URL https://doi.org/10.1112/plms/s2-42.1.230.

van den Berg, R., Awh, E., and Ma, W. J.
Factorial comparison of working memory models.
Psychological Review, 121(1):124–149, 2014.
doi: 10.1037/a0035234.
URL https://doi.org/10.1037/a0035234.

Van Hasselt, H., Guez, A., and Silver, D.
Deep reinforcement learning with double q-learning.
In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B.
Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm, July 2017.
URL http://arxiv.org/abs/1610.05729.
arXiv:1610.05729 [cs].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I.
Attention is all you need, 2017.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S.
Entity abstraction in visual model-based reinforcement learning.
In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Vilares, D., Strzyz, M., Søgaard, A., and Gómez-Rodríguez, C.
Parsing as pretraining.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 9114–9121, 2020.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D.
Grandmaster level in StarCraft II using multi-agent reinforcement learning.
Nature, 575(7782):350–354, October 2019.
doi: 10.1038/s41586-019-1724-z.
URL https://doi.org/10.1038/s41586-019-1724-z.

Vygotsky, L.
The Collected Works of L. S. Vygotsky.
Springer US, 1988.
doi: 10.1007/978-1-4613-1655-8.
URL https://doi.org/10.1007/978-1-4613-1655-8.

Vygotsky, L. S.
Thought and Language.
MIT Press, London, England, August 1965.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A.
Voyager: An Open-Ended Embodied Agent with Large Language Models, May 2023a.
URL http://arxiv.org/abs/2305.16291.
arXiv:2305.16291 [cs].

Wang, R., Lehman, J., Clune, J., and Stanley, K. O.
Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions.
arXiv preprint arXiv:1901.01753, 2019.

Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., and Stanley, K.
Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions.
In *International Conference on Machine Learning*, pp. 9940–9951. PMLR, 2020.

Wang, R., Jansen, P., Côté, M.-A., and Ammanabrolu, P.
Scienceworld: Is your agent smarter than a 5th grader?
In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, December 2022.

Wang, Y., Le, H., Gotmare, A. D., Bui, N. D. Q., Li, J., and Hoi, S. C. H.
CodeT5+: Open Code Large Language Models for Code Understanding and Generation, May 2023b.
URL http://arxiv.org/abs/2305.07922.
arXiv:2305.07922 [cs].

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N.
Dueling network architectures for deep reinforcement learning.
In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Wang, Z., Cai, S., Liu, A., Ma, X., and Liang, Y.
Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents.
arXiv preprint arXiv:2302.01560, 2023c.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V.
Unsupervised control through non-parametric discriminative rewards.
arXiv preprint arXiv:1811.11359, 2018.

Wasserman, E. A., Castro, L., and Freeman, J. H.
Same–different categorization in rats.
Learning Memory, 19(4):142–145, 2012.
doi: 10.1101/lm.025437.111.
URL http://learnmem.cshlp.org/content/19/4/142.abstract.

Waxman, S. R. and Markow, D. B.
Words as invitations to form categories: Evidence from 12-to 13-month-old infants.
Cognitive psychology, 29(3):257–302, 1995.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., brian ichter, Xia, F., Chi, E. H., Le, Q. V., and Zhou, D.
Chain of thought prompting elicits reasoning in large language models.
In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022a.
URL https://openreview.net/forum?id=_VjQlMeSB_J.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D.
Chain-of-thought prompting elicits reasoning in large language models.
In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022b.

URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

White, R. W.
Motivation reconsidered: The concept of competence.
Psychological Review, 66(5):297–333, sep 1959.
doi: 10.1037/h0040934.
URL https://doi.org/10.1037%2Fh0040934.

Whitehead, A. N. and Russell, B.
Principia Mathematica 3 Volume Set.
Cambridge University Press, Cambridge, England, January 1923.

Wiener, N.
Cybernetics.
The MIT Press. MIT Press, London, England, 2 edition, January 1961.

Winograd, T.
Procedures as a representation for data in a computer program for understanding natural language.
1971.
URL https://api.semanticscholar.org/CorpusID:54114373.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., and Funtowicz, M.
Transformers: State-of-the-art natural language processing.
In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Xiao, T., Chan, H., Sermanet, P., Wahid, A., Brohan, A., Hausman, K., Levine, S., and Tompson, J.
Robotic skill acquisition via instruction augmentation with vision-language models.
arXiv preprint arXiv:2211.11736, 2022.

Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., and Jiang, D.
Wizardlm: Empowering large language models to follow complex instructions.
arXiv preprint arXiv:2304.12244, 2023.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S.
How powerful are graph neural networks?
CoRR, abs/1810.00826, 2018.
URL http://arxiv.org/abs/1810.00826.

Yang, J., Prabhakar, A., Narasimhan, K., and Yao, S.
Intercode: Standardizing and benchmarking interactive coding with execution feedback, 2023.

Yao, S., Rao, R., Hausknecht, M., and Narasimhan, K.
Keep CALM and explore: Language models for action generation in text-based games.
In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8736–8754, Online, November 2020. Association for Computational Linguistics.
doi: 10.18653/v1/2020.emnlp-main.704.
URL https://aclanthology.org/2020.emnlp-main.704.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K.
Tree of thoughts: Deliberate problem solving with large language models.
arXiv preprint arXiv:2305.10601, 2023.

Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A., and Tenenbaum, J. B.
Clevrer: Collision events for video representation and reasoning.
arXiv preprint arXiv:1910.01442, 2019.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J.
Deep sets.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
URL https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N.
Star: Bootstrapping reasoning with reasoning.
Advances in Neural Information Processing Systems, 35:15476–15488, 2022.

Zhang, J., Lehman, J., Stanley, K., and Clune, J.
OMNI: Open-endedness via Models of human Notions of Interestingness, June 2023.
URL http://arxiv.org/abs/2306.01711.
arXiv:2306.01711 [cs].

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al.
Opt: Open pre-trained transformer language models.
arXiv preprint arXiv:2205.01068, 2022.

Zhou, H., Kadav, A., Lai, F., Niculescu-Mizil, A., Min, M. R., Kapadia, M., and Graf, H. P.
Hopper: Multi-hop transformer for spatiotemporal reasoning.
In *International Conference on Learning Representations*, 2021.
URL https://openreview.net/forum?id=MaZFq7bJif7.

Zhou, L. and Small, K.
Inverse reinforcement learning with natural language goals, 2020.

Zwaan, R. A. and Madden, C. J.
Embodied Sentence Comprehension, pp. 224–245.
Cambridge University Press, 2005.
doi: 10.1017/CBO9780511499968.010.

List of Figures

List of Tables