Intelligent Agents for Data Exploration (Author's Copy)

Sihem Amer-Yahia

Intelligent Agents for Data Exploration (Author's Copy)

Sihem Amer-Yahia
CNRS, Univ. Grenoble Alpes
sihem.amer-yahia@cnrs.fr

ABSTRACT

Data Exploration is an incremental process that helps users express what they want through a conversation with the data. Reinforcement Learning (RL) is one of the most notable approaches to automate data exploration and several solutions have been proposed. We first summarize some RL solutions that were built for different applications. In this context, various data exploration operators are leveraged including traditional roll-up and drill-down operations and text-based operations. An RL agent is trained to generate the best policy according to a hand-crafted reward function.

The benefit of training RL policies for specific data exploration tasks has been demonstrated more than once for exploring finding a needle in a haystack, for serendipitous galaxy exploration, for helping a customer land on a satisfactory product, for helping a conference chair build a program committee in a stepwise fashion, for summarizing large datasets, etc.

With the advent of Large Language Models and their ability to reason sequentially, it has become legitimate to ask the question: would LLMs and AI planning outperform an RL policy in data exploration? More specifically, would LLMs help circumvent retraining for new tasks and striking a balance between specificity and generality? This led us to designing LLM-powered approaches that introduce a new way of thinking about data exploration.

PVLDB Reference Format:
Sihem Amer-Yahia. Intelligent Agents for Data Exploration. PVLDB, 17(12): 4521-4530, 2024.
doi:10.14778/3685800.3685913

1 DATA EXPLORATION

Data Exploration is the process of interacting with data to unveil its content. It is incremental by nature because it should invite the user to express what they want through a conversation with the data. The ability to find useful data depends on two aspects: (1) *data and scenario complexity*: searching for individual records ranges from looking for "a needle in a haystack" to "scattered deposits" of clustered records, and (2) *human ability*: some users may be very familiar with the data, the tools, or the application domain, and know how and where to search, others may not be able to express their needs. Providing users with some automation to help them uncover what the data contains is key to an effective and successful exploration.

Automation is hard because full automation runs the risk of encapsulating unwanted semantics, and little automation leaves the burden of navigating in the data to users. Finding the right balance is ill-defined and remains an unsolved question. This paper explores the spectrum of automating data exploration with agents ranging from intentionally constrained agents trained with Reinforcement Learning (RL) to very free agents, a.k.a. Large Language Models (LLMs). It ends with a discussion of new opportunities.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685913

1.1 Use Cases

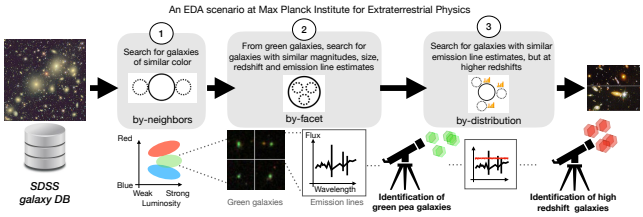

Figure 1: Exploring The Sloan Digital Sky Survey (SDSS) Data

Exploring Galaxies. Today's astrophysicists spend considerable time writing and reformulating SQL queries over the SDSS SkyServer database[1]. Many discoveries happen "accidentally". That was the case for the discovery of Green Pea galaxies that recently gained attention in Astronomy as one of the potential sources that drove cosmic reionization. What astronomers need is the ability to search for subsets or supersets of objects (akin to drill-downs and roll-ups), and search for galaxies with similar properties or similar property distributions.

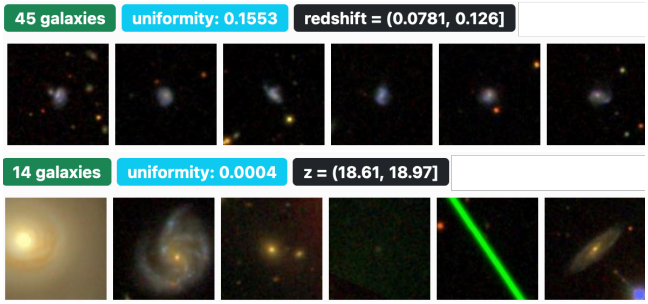

Figure 2: Examples of Uniform (top) and Non-uniform (bottom) Galaxy Itemsets.

Take for instance the case of Sri, an astronomer who wants to engage in finding as many Green Peas as possible, and possibly other kinds of galaxies that are similar. She needs to explore the data and refine her needs on-the-go. Figure 1 shows a sequence of three steps that form an automated exploration pipeline. The pipeline starts looking for *familiar yet different objects*: it first *finds*

[1]See the query interface and logs produced here http://skyserver.sdss.org/log/en/traffic/sql.asp

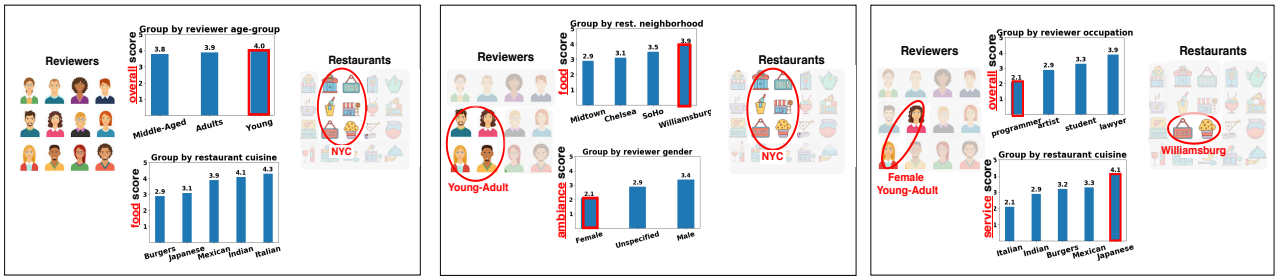

Figure 3: Example of a three-step exploration. The user iteratively examines subsets of the reviewer and item tables. Links between selected reviewer and item groups are aggregated as rating maps, showing the most "interesting" trends in the data.

neighboring objects, with similar colors as the Green Pea galaxies and then it *breaks down those objects into subsets of similar spectral properties.* These two steps result in more Green Peas. At this stage, exploration becomes adventurous and *returns different objects with comparable distributions of relative ratios and strength of emission lines at higher redshifts.* As a result, Sri discovers that Green pea galaxies are analogous to high redshift galaxies.

Consider now Max who is looking to summarize galaxy data. A summary can be defined as *a diverse set of k sets of items each of which is uniform, i.e., it contains items that are similar to each other.* Itemsets are different from each other, leading to a diverse summary. Figure 2 shows examples of uniform and non-uniform itemsets of galaxies derived from SDSS. One can see that uniform itemsets are easier to interpret by humans. Users would benefit from a multi-shot summarization approach that delivers k uniform and novel itemsets at each step until the full data has been explored.

Exploring Subjective Data. Exploring data on the social Web is a common and tedious task that people rely on for decision making. Consider for instance Mary who is looking for insights on restaurants in New York City. Figure 3 summarizes a 3-step exploration. Mary first examines the *reviewers' overall ratings* and sees no significant difference between age groups (upper histogram). As a young adult, her next operation is to *look deeper into that group* in Step II. She discovers that they gave the highest ratings for restaurants in Williamsburg (upper histogram). She also finds that on average, young women have given the lowest ambiance rating (lower histogram). In Step III, Mary *dives deeper into the ratings of young women*, and finds that programmers among them provided the lowest overall ratings (upper histogram). She also sees that those reviewers gave the highest service ratings to Japanese restaurants (lower histogram). With only a few steps, Mary obtains detailed insights on the opinions of people she relates to.

Consider now the case of Maya who wants to buy a high-quality camera for his upcoming nature photo-shooting. *Maya cannot form a precise query to express her need, as she's unsure what attributes matter in his purchase. She needs to cherry-pick items and reviews to make an informed decision.* Maya starts browsing cameras in Amazon that have a high optical view and processing power. Figure 4 depicts some exploration steps. Maya reviews the suggestions and focuses on a Fujifilm X100V camera. The system then suggests a representative set of reviews for the selected camera. This helps Maya get a better understanding of the pros and cons of that product, such as improved lens resolution and fragility of the camera. A review on "autofocus issues" attracts her attention, as this is important for capturing photos in nature. She intervenes and asks the system for other reviews on the same topic. In the results, Maya notices a review that suggests Canon cameras as better options for autofocus. She selects that review and the system recommends a few cameras related to that review among which Maya discovers a Canon 6D as her final target.

In our examples, users must visit a large space of objects to make an informed decision. Sometimes they need to refine their queries or draw insights from unstructured text. This process is tedious and prone to noise as it relies on one's technical abilities and domain knowledge.

1.2 Exploration Model

An exploration session is a sequence of iterations obtained by applying a generic exploration function defined as follows:

$$explore(X, \mathcal{X}, k) \rightarrow 2^{\mathcal{X}} \tag{1}$$

The exploration function encapsulates the semantics of various exploration operators. It admits a set of object $X \in \mathcal{X}$ that may be a set of galaxies, products, reviews, and returns k subsets of $\mathcal{X}$ that act as input to the next iteration. For instance, $explore(X_1, \mathcal{X}, 5)$ may return 5 subsets or supersets of X_1, or the 5 most similar sets of objects to X_1.

2 LEVERAGING REINFORCEMENT LEARNING

Recent work in automating data exploration has led to frameworks that leverage Reinforcement Learning (RL) to enhance the user experience [3, 6, 14, 16, 17, 20]. These frameworks rely on the formalization of a Markov Decision Process (MDP) that captures states representing subsets of the data being explored, actions representing operators used to navigate between states, transition probabilities, and a reward function used to train an agent and produce an optimal data exploration policy.

2.1 Modeling Exploration Processes

An exploration process is often modeled as an MDP comprising a quadruple (S, E, P, R) where:

- S is a discrete set of exploration states;
- E is a set of exploration actions, where each action instantiates our function $explore(X, \mathcal{X}, k)$ and enables a transition between consecutive exploration states s_t and s_{t+1};

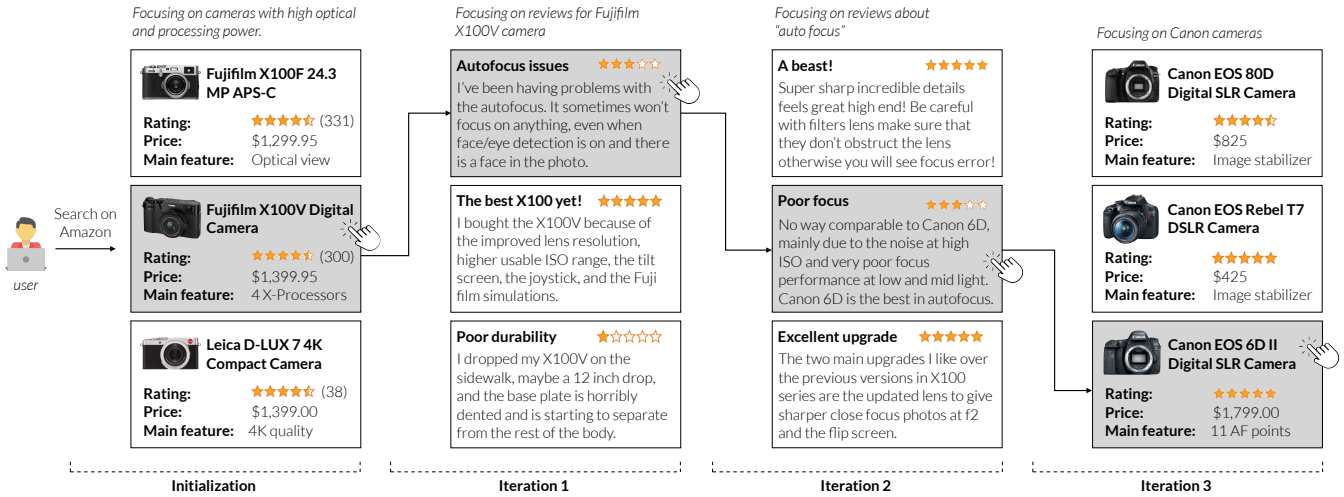

Figure 4: Example of review-based item exploration.

- $P(s_{t+1}|s_t, e_t)$ are the probabilities that the exploration action e_t will change state $s_t \in S$ to state $s_{t+1} \in S$;
- $R(s_{t+1}|s_t, e_t)$ are rewards for transitioning from state $s_t \in S$ to $s_{t+1} \in S$ by applying the exploration action e_t.

The exploration state s_{t+1} is obtained by applying action e_t to a previous state s_t. The exploration process goes on by selecting another action e_{t+1}. The probabilities $P(s_{t+1}|s_t, e_t)$ reflect the behavior of the environment when exploration actions are applied, i.e., they represent what is displayed to the user in iteration $t+1$, if e_t is applied. Usually, a *model-free* setting [23] is considered, where the probabilities are not known a priori, and depend on a reward function. An *exploration policy* π is a mapping function from an exploration state s_t to an action e_t, where $\pi(s_t) = e_t$. An exploration session generated by π is $\mathcal{S}^\pi = [(s_1, \pi(s_1)), \ldots, (s_n, \pi(s_n))]$.

The MDP is used to learn a policy of an agent with the goal to maximize its expected reward such as:

$$\pi^* = argmax_\pi E[R^\pi] \quad (2)$$

where π is a policy and R^π is the reward obtained by π.

The choice of actions and reward is dictated by an application. We provide an overview of different systems and their implementation choices for the MDP states and rewards.

2.2 DORA The Explorer

DORA THE EXPLORER was developed to explore very large galaxy data in SDSS [16, 17]. Table 1 shows the operators in DORA THE EXPLORER. The second column depicts their equivalent definition in the *Region Connection Calculus 8* (RCC8) formalism [9, 19].

DORA THE EXPLORER defines a target set of familiar objects T and a reward function that is expressed as *a linear combination of familiarity and curiosity*. Familiarity captures the proportion of target objects found at each step [3, 11, 20]. This definition is insufficient when target objects are "drowned" in very big sets such as galaxies. Hence, the need to balance (extrinsic) familiarity and (intrinsic) curiosity with the goal of rewarding newly encountered states similarly to learning to play a game [15].

Table 1: Exploration operators. The initial data is represented with a bold line and the destination results are represented with dashed lines.

Operator	RCC8 Formalism [19]	Output description
by-facet(D, A)	NTPPi	returns as many subsets of D as there are combinations of values of attributes in A
by-superset(D, k)	NTPP	returns the k smallest supersets of input set D (k is application-dependent)
by-distribution(D)	DC	returns all sets that are distinct from the input set D and whose attribute value distribution is similar to D
by-neighbors(D, a)	EC	returns 2 sets that are distinct from the input set D and that have the previous (smaller) and next (larger) values for attribute a

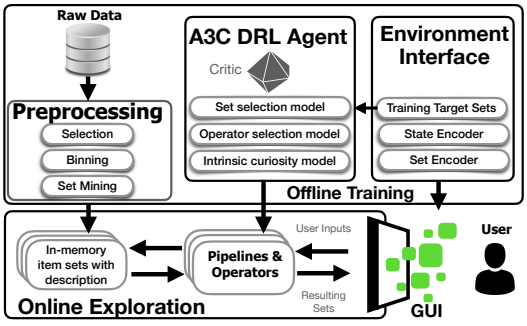

Figure 5: Architecture of DORA THE EXPLORER [17].

DORA THE EXPLORER's architecture is given in Figure 5. Data is pre-processed to instantiate a set-based data model. Equi-depth binning is applied to numerical attributes and the LCM closed frequent pattern mining algorithm [27] generates itemsets. Model

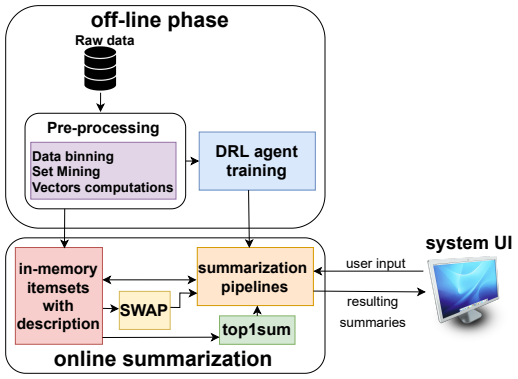

Figure 6: Architecture of EDA4Sum for multi-step summarization [31].

training relies on a Tensorflow-based implementation of A3C[2], a parallelized and asynchronous Deep RL (DRL) architecture: multiple actor-learners are dispatched to separate instances of the environment within which they interact, collect experiences and asynchronously push their gradient updates to a central target network.

The combination of input set, operators, and parameters, yields a large action space. That is handled by leveraging two actors. The first actor selects on which set the next operator should be executed and the second selects the operator and parameters. The states the agent goes through are evaluated by a single critic model, producing the advantages used to train both actors. The DRL agent is trained with the environment parameters: (1) the *Target Set*; (2) a *Set Encoder* that generates a feature vector for each set of objects. The features are the set size, the set description, the number of distinct values in the set, and the entropy of the values in the set, and (3) a *State Encoder* that concatenates the encoded vectors of every displayed set. For each training, the system chooses different weights for intrinsic and extrinsic rewards. The outcome of training is saved in a storage unit that contains several pre-trained models.

2.3 EDA4SUM

EDA4Sum was developed for multi-step large data summarization [18, 31]. Its data model is based on DORA THE EXPLORER and its reward is formulated as *a weighted linear combination of uniformity, diversity, and novelty of summaries* produced at each step.

Figure 6 depicts the architecture of EDA4Sum. The offline phase is based on DORA THE EXPLORER [17] and is used to train multiple DRL agents, each of which has different weights in the reward, to summarize data. EDA4Sum generates summarization pipelines following one of three modes: *Manual*, *Partial Guidance* and *Full Guidance*. Pipeline execution starts with the SWAP algorithm [32] that finds the k most uniform and diverse itemsets. The next steps are chosen by pre-trained agents.

The UI of EDA4Sum is depicted in Figure 7 (with the SPOTIFY music dataset that includes 232K music tracks with 11 attributes). The current 7-step pipeline is shown in the A zone and its results in the B zone. The user investigates a dataset by specifying the summarization mode, underlying agent, and possibly customizing the weights of uniformity, diversity and novelty (C zone). In *Manual* and *Partial* modes, the user can override the next operator and its parameters (D zone). The user may want to store the current pipeline or upload a previously stored one to execute it (E zone).

[2]https://github.com/marload/DeepRL-TensorFlow2/

Table 2: Review-based state-action features in GUIDES

Feature group	Components
Input object x_t	object type (item or review, one feature), rating (one feature for each 5-star rating scale), text word count (one-hot), tag count (one-hot), and sentiments and topics (10 features for each)
Output objects X_t	Same features as the ones for x_t, for each output object $x \in X_t$
Exploration action e_t	$textdiv(X_t)$, $numdiv(X_t)$, $coverage(X_t)$, all one-hot encoded
Target $\mathcal{T}$	number of targets discovered in $\mathcal{T}$ in one-hot encoding

2.4 GUIDES

GUIDES was developed to help users make informed decisions on e-commerce sites [17]. The set of objects it manipulates are products and reviews. The reward of a state $R(s_{t+1}|s_t, e_t)$ reflects the utility of X_{t+1}. A positive reward is received every time some objects belonging to a target set of products and reviews $\mathcal{T}$ are discovered. To capture that, GUIDES uses a *Reward Machine* [7] with three types of rewards: neutral, target, and similarity. If a set of domain-dependent rules Ω is violated in s_{t+1}, a *neutral reward* (e.g., 0) is given. If Ω holds and some of the target objects exist in X_{t+1}, a *target reward* (e.g., 1) is given. If Ω holds but no target is reached, a *similarity reward* is given proportionally to the degree of progression towards $\mathcal{T}$:

$$R(s_{t+1}|s_t, e_t) = \begin{cases} [0]^{\mathcal{K}+1} & \text{if } \Omega(s_{t+1}) = \mathit{False} \\ [|X_{t+1} \cap \mathcal{T}|]^{\mathcal{K}+1} & \text{else if } X_{t+1} \cap \mathcal{T} \neq \emptyset \\ \mathit{utility}(X_{t+1}, \mathcal{T}) & \text{otherwise} \end{cases} \quad (3)$$

Reward is *multi-valued* because the utility function outputs a vector of size $\mathcal{K}+1$. This is a natural representation as the textual content is identified with different exploration dimensions. For instance, a review might have high utility in terms of its topic dimension and a low utility when it comes to the sentiment dimension. This means reward maximization becomes a multi-objective optimization problem, where each reward dimension acts as one optimization objective.

GUIDES represents states and actions jointly in a fine-granularity space by leveraging a set of data-dependent state-action linear features $ft(s, e)$. These features enable the discovery of commonalities in the state-action space and hence widen the scope of the learned policy. Table 2 provides details of the features. GUIDES leverages approximate control methods [23] to learn an approximation of the action-value function $\hat{Q}(\mathbf{w}, s, e) \approx Q(s, e)$ where $\mathbf{w} \in \mathbb{R}^m$ and m is the number of features, $m \ll |S \times E|$. We compute the approximated action-value function as $\hat{Q}(\mathbf{w}, s, e) = \mathbf{w}^\top ft(s, e)$. The weights $\mathbf{w}$ are learned by the prediction network θ.

Figure 8 provides the overall architecture of GUIDES. The offline training phase simulates a DRL agent to learn a policy. The agent interacts with a prediction neural network (step 1) to update its weights by performing actions on the environment (step 2). GUIDES is equipped with a multi-valued reward mechanism that filters actions with dominated rewards (step 3). These rewards are then

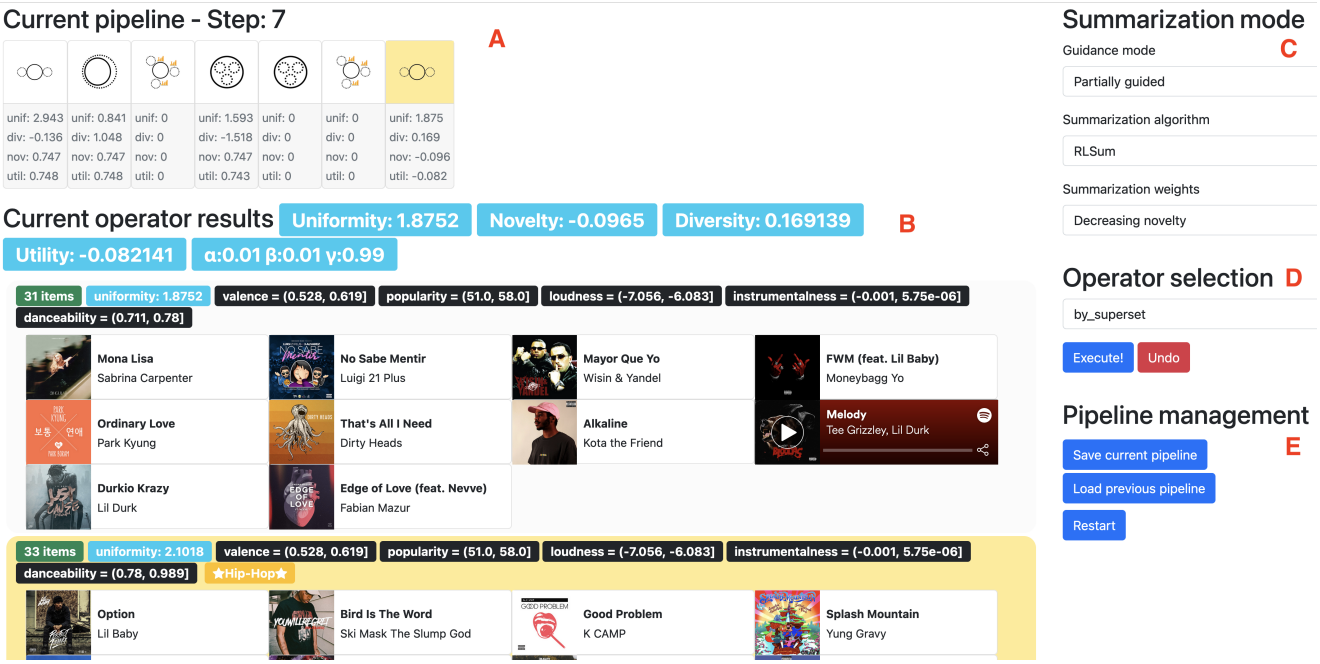

Figure 7: UI of EDA4Sum with SPOTIFY.

used (step 4) to optimize the policy (step 5). The optimal policy will then be leveraged in an online context to recommend actions to the user for effective decision-making.

GUIDES leverages a Deep RL (DRL) algorithm called Deep Q-Networks (DQN) [13] as the learning component of GUIDES. DQN was chosen among a variety of discrete-space DRL algorithms because its architecture complies with compound targets. That means that the target is spread throughout the exploration session. GUIDES' algorithm starts by initializing a buffer $\mathcal{B}$ and two neural networks, the prediction network θ and the target network θ^T, and then proceeds by updating them. It outputs an optimized exploration policy which is represented in terms of θ's learned weights.

In DQN, the updates do not impact the neural network weights right after each interaction with the environment. For more stability in the training, each interaction is first pushed to a buffer $\mathcal{B}$, and then training is performed by sampling from $\mathcal{B}$. Each interaction stored in $\mathcal{B}$ is a tuple $\omega_t = \langle s_t, e_t, r_t, s_{t+1} \rangle$, an *experience tuple* at iteration t, and the sampling process is an *experience replay*.

The prediction network θ is trained (using experience replays from $\mathcal{B}$) as a function approximator for the action-value function, denoted as $Q(s, e; \theta)$. To achieve learning stability, the parameters of the prediction network are copied to the target network θ^T, every ζ steps, where ζ is a hyper-parameter. The internal architecture of both prediction and target networks are often identical. In GUIDES, the architecture used both networks consists of four layers of linear transformation with width 1024 and rectifier non-linearity functions (ReLU) in-between, and $\zeta = 100$. It also uses Adam [8] as the optimizer with a learning rate $\alpha = 0.0003$.

Each step in the learning process begins by initializing a starting state $s_t = \langle x_0, X_0 \rangle$, picking an exploration action $e_t = get\_action(\varepsilon)$ using the ε-greedy method, and then following an ensuing exploration session until its termination. In each step of the session, first

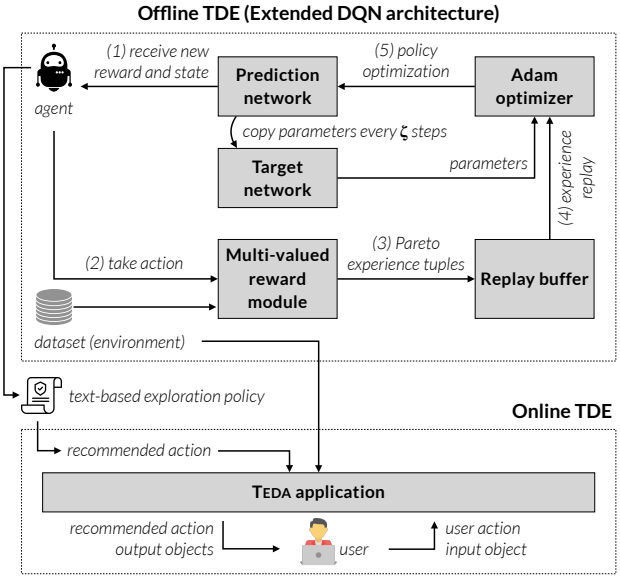

Figure 8: Architecture of GUIDES [14] for Text-based Data Exploration (TDE).

the chosen action is applied to obtain the reward r_t and the next state s_{t+1}, and then the experience tuple is formed to be pushed to the buffer. Following that, an experience tuple $\bar{\omega}$ is replayed from $\mathcal{B}$ to update the parameters of the Q function approximator θ.

The DQN architecture is modified to account for the multi-objective reward. Experience tuples are pushed into the buffer iff they are *not dominated* by any other tuple, which means they belong to the Pareto front [25]. This ensures that the buffer always contains non-dominated experiences.

3 LLMS: A NEW ERA FOR DATA EXPLORATION?

LLMs have demonstrated impressive capabilities in natural language understanding and generation, and they now extend their capabilities beyond language tasks. A recent trend leverages LLMs for complex tasks [12]. This makes LLMs ideal candidates for automating data exploration tasks. While resolving complex tasks involves decision-making at each step, it also requires reasoning capabilities. Chain-of-Thought (CoT) [28] prompting has become a notable advancement in enhancing the reasoning capabilities of large LLMs. This method involves guiding LLMs to break down complex problems into intermediate steps, improving their ability to solve multi-step reasoning tasks. Language Agent Tree Search (LATS) [34] specifically addresses the limitations of LLMs in decision-making by adapting Monte Carlo Tree Search to help LLMs interact with an external environment, providing them with external feedback, and self-reflection for evaluating potential actions.

Multi-Agent Collaboration for LLMs. This approach leverages teams of specialized LLMs to tackle complex tasks [24]. Each LLM agent can be specialized for specific tasks and an intelligent communication and collaboration (task division, information sharing) enables efficient problem solving. The open-source framework AutoGen [29] exemplifies that by allowing to create applications with multiple interacting LLM agents. Systems like LangChain [1] further facilitate the coordination of multiple LLMs through LangGraph [2], a specialized map for agent coordination. Each LLM agent in LangChain can be customized with unique instructions and access to specific tools. This is illustrated in AutoDev [26] that uses a combination of agents and tools to facilitate automated software development tasks.

We ask the question of whether we can leverage the decision-making capabilities of LLMs to manage LLM agents, orchestrating their collaboration for efficient data exploration using LangGraph.

Enhancing LLMs with RL for Data Exploration. One of the concerns with leveraging LLMs for data exploration is the need to constrain general-purpose LLMs appropriately and avoid hallucinations. RL can play an important role in enhancing LLMs in that direction. The REMEMBERER framework enhances LLMs with an external memory component [33]. It integrates an RL-powered memory with an LLM, enabling decision-making based on past experiences. REMEMBERER's memory is updated through RL, yielding high rewards. This allows the LLM to leverage past experiences without the need for fine-tuning.

Enhancing RL with LLMs for Data Exploration. A critical aspect of RL is defining rewards, where LLMs can play a role as a self-refining reward function [22]. For instance, frameworks like Eureka [10] show the capability of LLMs to achieve human-level performance in designing reward systems. By dynamically adapting the reward function based on feedback, Eureka allows RL agents to learn more effectively in complex environments. Additionally, methods like ELLM [5] explore the use of LLMs to guide the pre-training phase of RL. In this approach, LLMs suggest goals for the agent based on its current state, and the agent is then rewarded for achieving these LLM-suggested goals. This highlights the potential of LLMs to not only design rewards but to also directly influence the exploration process within RL.

We ask the question of whether we can blend RL and LLMs effectively to benefit from the wide knowledge and power of LLMs to specify actions and rewards while leveraging RL to constrain LLM hallucinations.

4 INTELLIGENT AGENTS FOR DATA EXPLORATION

In an attempt to address the questions asked in the previous section, we designed an environment and four agents for data exploration: *RL, RawLLM, QValLLM, FeedbackLLM.* These agents are used to explore products and reviews on Amazon. The variation between LLM-enabled agents lies in how the supervisor determines which agent will act next, each method representing a unique decision-making strategy. Our four agents yield different ways for the decision-making process within the workflow, each leveraging different sources of information to guide the selection of the next agent.

4.1 The Environment

We have a set of objects O. We use an Amazon dataset where each object (product or review) has features such as tags, attributes, sentiment, summary, and text.

A state is the result of an action, so it is defined as an input object in O, a set of k output objects in O and an indicator of the last action used. In practice this can be just ids. We can write $\mathbb{S} = O^{1+k} \times (id_a)_{a \in \mathbb{A}}$.

An action consists in picking an agent in $(\mathcal{L}_m)_{m \in [1,n]}$ and the id of an object in the state, and using this agent to get a new list of objects. For each agent we have a prompt-builder function $Prompt_m : \mathbb{S} \to \mathbb{T}$ (which can be the same for all agents), and a state-decoder function $decode : \mathbb{T} \to \mathbb{S}$ to transform the agent outputs into a state (assuming that the agents' output is a text). The result of action m in state s is simply $s' = decode(\mathcal{L}_m(Prompt_m(S)))$.

Actions are defined over object features. Specifically, the tag matcher retrieves the top 10 objects with the highest similarity based on tags compared to an input object, while the attribute matcher operates similarly but with attributes, and so forth. All similarities are pre-computed. We use Voyage AI to build embeddings for our objects. Pairwise object similarities on text, sentiment, attributes, tags and summaries, are pre-computed on with cosine over the embedding vectors of the objects.

The reward function is defined based on a predefined target set of objects T. We define R_T as the Jaccard similarity between the set of objects present in the states and the target set: $R_T((s_0, s_1, ..., s_k)) = J(\{s_i | i \in [0, k]\}, T)$.

The environment we introduced allows us to define one RL agent and several *multi-agent LLMs.* The RL agent reflects a constrained agent trained for a specific application. The multi-agent LLM workflow uses LangChain [1] to coordinate multiple LLMs through LangGraph [2]. They define an LLM orchestrator/supervisor responsible for managing exploration and task logic, and multiple LLM agents, one for each action. In LangChain, each agent is assigned to a particular exploration action. The supervisor LLM is in charge of selecting the appropriate agent for the next action. The

agents reflect the tag matcher, attribute matcher, text matcher, summary matcher, and sentiment matcher. he tools are instrumental and are used by the LLM agents to perform various tasks within the defined workflow. We implemented five custom tools, each corresponding to a specific agent. These tools are functions that the agent knows when to call according to the context. The first tool is SimilarityByTag, the second one SimilarityByAttribute, the third one SimilarityBySummary, the fourth one SimilarityByText and the last one SimilarityBySentiment. Each tool admits one object and generates the 10 most similar objects.

The supervisor decides which agent starts the process. Then, the selected agent gathers 10 objects. Based on this outcome and its parametric knowledge, the supervisor decides whether to continue with the next agent or to conclude the workflow and report the findings. This step-by-step approach ensures that each agent's results guide the next steps in the process.

4.2 The Agents

RLAgent is defined with a trained Q-value function $Q : \mathbb{S} \to (\mathbb{A} \to \mathbb{R})$ that assigns to each state a value for each action possible at that state. This Q-function is trained to approximate the value function $V*$, that verifies the Bellman equation:

$$V^*(s) = \sum_{s' \in \mathbb{S}} \mathbb{P}(s'|s, \pi(s)) \left[R(s') + \gamma V^*(s')\right]$$

so that

$$Q(s, a) \approx \sum_{s' \in \mathbb{S}} \mathbb{P}(s'|s, a) V^*(s')$$

A policy $\pi : \mathbb{S} \to \mathbb{A}$ is trained to always pick the action with the highest estimated value: $\pi(s) = argmax_{a \in \mathbb{A}} Q(s, a)$. Since the Q-function depends on the reward function, so does the agent.

The agent *RawLLM* is defined by a supervisor LLM $\mathcal{L}_{Sup}$, a prompt-builder function $Prompt : \mathbb{S} \to \mathbb{T}$, and an action-decoder function *Action*. The policy $\pi : \mathbb{S} \to \mathbb{A}$ is here defined as $\pi(s) = Action(\mathcal{L}_{Sup}(Prompt(S)))$. In this case the policy will be generted by the supervisor LLM who will decide which agents and tools to call to achieve what is asked of it.

The agent *QValLLM* is defined by a supervisor LLM $\mathcal{L}_{Sup} : \mathbb{T} \to \mathbb{T}$ (where $\mathbb{T}$ as the set of possible texts, which are finite sequences of tokens), a trained Q function, a prompt-builder function $Prompt_Q : \mathbb{S} \to \mathbb{T}$, and an action-decoder function $Action : \mathbb{T} \to \mathbb{A}$ mapping outputs to actions. The policy $\pi : \mathbb{S} \to \mathbb{A}$ is here defined as $\pi(s) = Action(\mathcal{L}_{Sup}(Prompt_Q(S)))$. Since the Q-value depends on the reward, so does the agent. The open question here is how to add information from the Q-function to the prompt; i.e., how to define $Prompt_Q$. A possibility would be to append the values associated to possible actions in state s according to Q.

The agent *FeedbackLLM* is defined by a feedback function imitating a reward $R^\star : \mathbb{S} \to \mathbb{R}$ (but unlike the reward, it is available at inference time), a supervisor LLM $\mathcal{L}_{Sup}$, a prompt-builder function $Prompt_{R^\star}$ and an action-decoder function *Action* mapping outputs to actions. The policy $\pi : \mathbb{S} \to \mathbb{A}$ is here defined as $\pi(s) = Action(\mathcal{L}_{Sup}(Prompt_{R^\star}(S)))$. Similarly to *QValLLM*, the open question is to define $Prompt_{R^\star}$. A possibility would be for $Prompt_{R^\star}$ to simulate the actions and compute the feedback of the obtained states, then append these results to the prompt.

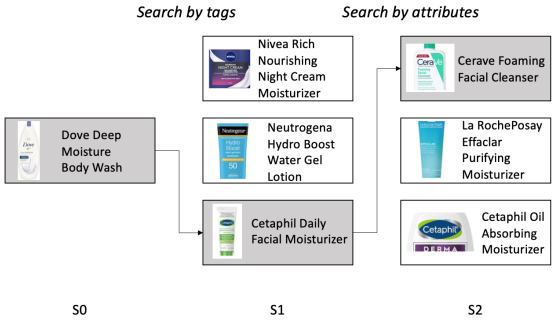

Figure 9: Exploration scenario of cosmetic products.

4.3 Prompt Construction

Both the agents and the environment require the construction of prompts based on a state and other parameters. The general methodology to build a prompt is to use a pre-defined instruction schema $I \in \mathbb{T}$ and a state descriptor $describe : \mathbb{S} \to \mathbb{T}$. *Prompt* is then defined by $Prompt(s) = I \cdot describe(s)$.

For *RawLLM* one does not need to go from text to state and then from state to text because all it needs is text (thus states could actually be defined as texts). But for *QValLLM* and *FeedbackLLM*, the state-decoder and the prompt-builder functions are applied.

In the case of *QValLLM*, the Q-table generated by *RL* is added to the prompt. One solution is to calculate the local value function $value_s : \mathbb{A} \to \mathbb{R}$ defined by $value_s(a) = Q(s, a)$, which would be fully described by an operator to_text. We then have $Prompt_Q(s) = I \cdot describe(s) \cdot to\_text(value_s)$ (to be more precise the instruction is completed with the following texts, which may not be necessarily concatenated at the end).

For *FeedbackLLM* we can do the same except that the local value function is defined based on the feedback function $R^\star$ by $value_s(a) = \sum_{s' \in \mathbb{S}} \mathbb{P}(s'|s, a) R^\star(s')$ (but in practice we would only do one call for each action with an MDP). Alternatively, if we want to avoid simulating all actions, we can also exploit the components of $R^\star$ on the local state. Let's suppose $R^\star$ is defined by a relevance function for individual objects r and a quality function on the whole state ϕ (measuring things like the diversity of the object covered), linearly combined in the following way: $R^\star(s) = \lambda_r \frac{1}{||O||} \sum_{o \in 0} r(o) + \lambda_\phi \phi(s)$ (were λ_r and λ_ϕ are real coefficients). Taking r_s as the function r restricted to the objects of the current state, $Prompt_{R^\star}$ could be defined by $Prompt_{R^\star} = I \cdot describe(s) \cdot to\_text(r_s) \cdot \phi(s)$.

4.4 Intelligent Agents in Action

We illustrate an exploration scenario in Figure 9 where Marie aims to identify cosmetic products suitable for oily skin. A predefined set of products and reviews from Amazon, aligning with Marie's criteria, was established. Subsequently, *RLAgent* trains a policy using this target set, containing 50 cosmetic products suitable for oily skin. *RL* also outputs a Q-table.

RL receives an initial state s_0, containing a single randomly chosen product from Amazon, the *Dove Deep Moisture Body Wash*. The agent uses its Q-function to assess available actions. At stage 0, the agent selects "search by tag similarity". Thus, the agent explores the database, identifying products with similar tags to the initial object.

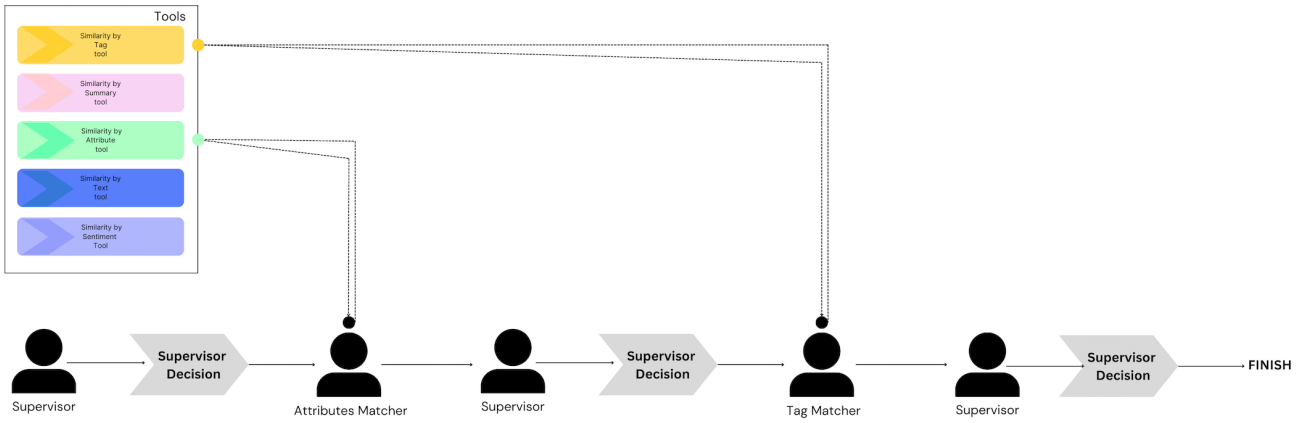

Figure 10: Example of a LanGraph workflow (*RawLLM*): The supervisor calls the Attributes Matcher. This agent calls the SimilarityByAttributes tool, and reports its findings to the Supervisor, that decides to call the Tag Matcher. This agent calls the SimilarityByTag tool, and reports its findings to the supervisor that decides to end the workflow.

Guided by its learned policy towards the target set, the agent updates the state s_1, to incorporate these products and the previously selected action. Using both the updated state s_1 and Q-function results, the agent selects the "search by attributes" action, leading to the final state s_2.

RawLLM begins with the same initial product as *RL*. However, the supervisor does not rely on any contextual information to choose the next action. It just uses its parametric knowledge and knows that its purpose is explore data to reach a user's information need (finding face creams for oily skin in the example). Figure 10 depicts the workflow of *RawLLM* implemented in LanGraph.

QValLLM receives the same initial state. The Q-function, trained with *RL*, evaluates all potential actions. At stage 0, "search by tag similarity" emerges as the action with the highest estimated value. This action triggers the utilization of a tag matcher LLM that accesses a database containing object tags extracted from both item descriptions and customer reviews. The tag matcher identifies 3 products sharing similar tags with the initial *Dove Cleanser* that can be seen in state s_1 in Figure 9. Among these products, *Nivea Rich Nourishing Night Cream Moisturizer* and *Neutrogena Hydro Boost Water Gel* are found to belong to the predefined target set for oily skin. The supervisor randomly selects *Cetaphil Daily Facial Moisturizer*, one of the retrieved products from the target set, and updates s_1, to include this product along with the other two retrieved products and the "search by tag similarity" action. Leveraging the updated s_1 and the Q-function values, the supervisor selects the next action, "search by attributes." This action activates the attribute matcher LLM, refining product recommendations and identifying three additional products, all belonging to the target set. With these new discoveries, the state transitions to s_2, encompassing the attribute matching action and the last chosen products: *Cerave Foaming Facial Cleanser*, *La Roche-Posay Effaclar Purifying Moisturizer*, and *Cetaphil Oil Absorbing Moisturizer*.

FeedbackLLM begins with the same initial product. However, instead of relying on the Q-function, all potential actions are evaluated by the supervisor LLM, that utilizes its own parametric knowledge and all the previous states. At s_0, the supervisor LLM selects the "search by tag similarity" action based on its acquired knowledge and the input product. *FeedbackLLM* then employs a tag matcher LLM to retrieve products, with the last two products belonging to the predefined target set (see Figure 9). The supervisor randomly selects *Cetaphil Daily Facial Moisturizer* and updates the state s_1, to include this product, the other two retrieved products, and the "search by tag similarity" action. What distinguishes *FeedbackLLM* from *QValLLM* is the incorporation of retrieved products belonging to the target set within the state. In the absence of specific training on the target set, a mechanism is necessary to guide the LLM towards it. By explicitly identifying the retrieved objects from the target set within the state, the model's focus on the target set is facilitated. Leveraging the updated state s_1 and the initial state s_0, the supervisor selects the next action, "search by attributes.", leading to the final state s_2.

4.5 Results and Analysis

We want to compare the performance of our agents based on the proportion of retrieved products that belong to a target set. We defined two target sets of 255 objects: the 5 best-rated computer products and their associated reviews ($T1$), and the 5 best-rated items from the photo and camera category with their associated reviews ($T2$). To evaluate the efficiency of different agent variants in discovering these target sets, we plotted the cumulative number of target items retrieved against the number of actions taken (iterations), ranging from 0 to 50 actions. Each experiment was conducted 10 times for both target sets and for each agent. The average performance was computed to provide a robust comparison.

The comparison between different target sets ensures that the agents are not only effective in a single context but can generalize their exploration strategies to diverse environments. This approach provides a comprehensive understanding of each agent's strengths and limitations, guiding the development of more adaptive and robust exploration algorithms.

Our results do not report *QValLLM* simply due to lack of time.

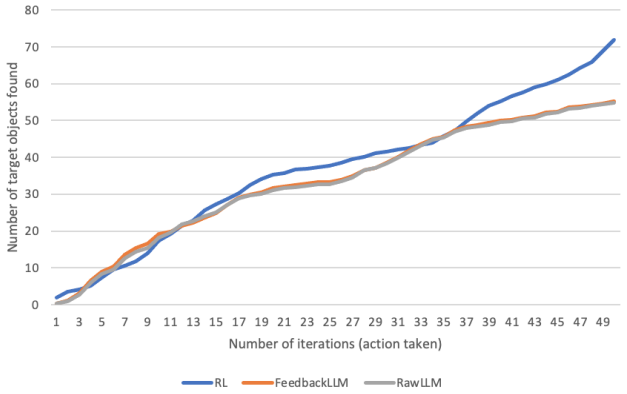

Figure 11: Cumulative number of target objects found by different agents across the number of actions (target set $T1$)

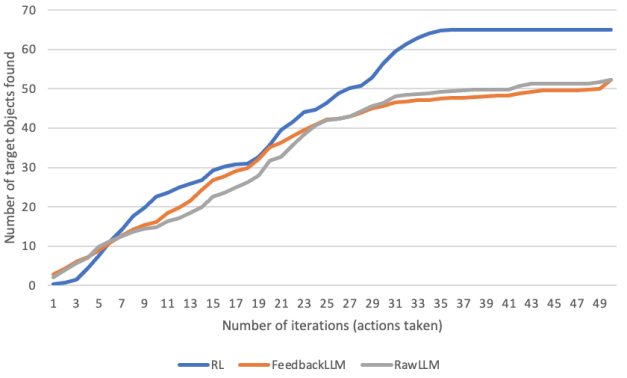

Figure 12: Cumulative number of target objects found by different agents across the number of actions (target set $T2$)

Figure 11 illustrates the results for the first target set $T1$ and 12 illustrates the results for the second target set $T2$. *RL* achieves the highest cumulative count of target objects retrieved consistently throughout the action sequence. *FeedbackLLM* performs comparably to *RawLLM*, maintaining a similar trajectory of target discovery, with both agents showing a slower increase compared to *RL*.

RL consistently outperforms the LLM-based agents across both target sets. The RL agent benefits from task-specific training (here, it is trained for this specific target set) which allows it to develop an optimal strategy for the exploration.

FeedbackLLM remains relatively close in performance to *RawLLM* across both target sets. It suggests that the feedback mechanism may need refinement or more substantial integration to significantly impact performance.

RawLLM maintains a steady rate of target discovery, similarly to *FeedbackLLM*. Its performance is consistent across different target sets, demonstrating its general exploration capability. As a pre-trained language model without task-specific training, *RawLLM* relies on its inherent knowledge and general exploration abilities. It performs adequately in discovering target items but lacks the task-specific optimization seen in *RL*.

The results from multiple target sets shows that *RL* is doing better due to its specialized training, which enables it to adapt and optimize its exploration strategies effectively for specific target sets. This leads to superior performance in discovering target items. Meanwhile, the LLM-based agents, *FeedbackLLM* and *RawLLM*, demonstrate good performance without specific training for exploration. However, the closeness of *FeedbackLLM* and *RawLLM* suggests that while feedback can enhance performance, its impact is limited and requires further investigation.

5 FUTURE DIRECTIONS

The comparison we provided of the agents developed in the previous section leads us to thinking that there is great potential for blending RL and LLMs to benefit from the background knowledge of LLMs and guide them through the exploration process without incurring training costs required for RL. Unlike RL that needs to be trained for each task, the training cost of an LLM can be amortized across a variety of exploration tasks.

The comparison we provided on different target sets emphasizes the trade-off between generalization and specialization. LLM-based agents, although not specifically trained for exploration, provide a reliable performance baseline that is consistent across varied scenarios. In contrast, *RL*'s task-specific training leads to higher performance in the trained contexts but requires retraining or adjustment in different environments. The limited performance boost from *FeedbackLLM* suggests the need to refine the feedback and subsequently the LLM. Further refinement in how feedback influences decision-making could enhance the agent's ability to adapt and improve its exploration strategies. This constitutes a new opportunity for leveraging new approaches for LLM refinement (e.g., [4, 30]).

Another aspect is validation that leverages human feedback. While Large Language Models (LLMs) are increasingly used to evaluate other LLMs, addressing their inherent biases and aligning them with human preferences remains a challenge. EvalGen [21] is a system that tackles this by combining user input with LLM capabilities. EvalGen iteratively refines evaluation criteria through a feedback loop: users define desired LLM outputs, EvalGen generates candidate evaluation methods (prompts or code), users grade outputs based on these methods, and EvalGen uses this feedback to improve the criteria. This approach acknowledges the subjective and iterative nature of human evaluation, particularly the phenomenon of "criteria drift". It refers to the phenomenon where the human evaluators' understanding of what constitutes a good output evolves as they interact with the system, influencing the evaluation criteria itself. EvalGen presents a promising direction for LLM evaluation assistants, underlining the importance of incorporating human judgment throughout the process.

There is a need for a principled evaluation framework for exploration policies by integrating user feedback and LLM capabilities. This would pave the way to an evaluation system that refines itself according to human preferences and feedback and truly accounts for user needs in exploratory data analysis.

ACKNOWLEDGEMENTS

I would like to thank my numerous collaborators in data exploration: Senjuti Basu Roy, Laure Berti-Equille, Maximilan Fabricius, Tova Milo, Behrooz Omidvar Tehrani, Aurélien Personnaz, Mariia Seleznova, Eric Simon, Srividya Subramanian, Brit Youngmann. I also thank my colleagues and students who provided valuable content and feedback on earlier versions of this paper: Laks V. S. Lakshmanan, Noha Ibrahim, Nassim Bouarour, Alecandre Rio, Charline Fiorentino, Cypren Michel-Deletie.

REFERENCES

[1] [n.d.]. *LangChain: https://www.langchain.com/.*

[2] [n.d.]. *LangGraph: https://python.langchain.com/v0.1/docs/langgraph/.* https://python.langchain.com/v0.1/docs/langgraph/

[3] Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically generating data exploration sessions using deep reinforcement learning. In *SIGMOD*. 1527–1537.

[4] Cheng-Han Chiang and Hung yi Lee. 2023. Can Large Language Models Be an Alternative to Human Evaluations? arXiv:2305.01937 [cs.CL] https://arxiv.org/abs/2305.01937

[5] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding Pretraining in Reinforcement Learning with Large Language Models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 8657–8677. https://proceedings.mlr.press/v202/du23f.html

[6] Ori Bar El, Tova Milo, and Amit Somech. 2019. ATENA: An Autonomous System for Data Exploration Based on Deep Reinforcement Learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, Wenwu Zhu, Dacheng Tao, Xueqi Cheng, Peng Cui, Elke A. Rundensteiner, David Carmel, Qi He, and Jeffrey Xu Yu (Eds.). ACM, 2873–2876. https://doi.org/10.1145/3357384.3357845

[7] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2107–2116.

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Sanjiang Li and Mingsheng Ying. 2003. Region connection calculus: Its models and composition table. *Artificial Intelligence* 145, 1-2 (2003), 121–146.

[10] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-Level Reward Design via Coding Large Language Models. *CoRR* abs/2310.12931 (2023). https://doi.org/10.48550/ARXIV.2310.12931 arXiv:2310.12931

[11] Patrick Marcel, Nicolas Labroche, and Panos Vassiliadis. 2019. Towards a benefit-based optimizer for Interactive Data Analysis. In *EDBT/ICDT*.

[12] Shervin Minaee, Tomás Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large Language Models: A Survey. *CoRR* abs/2402.06196 (2024). https://doi.org/10.48550/ARXIV.2402.06196 arXiv:2402.06196

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[14] Behrooz Omidvar-Tehrani, Aurélien Personnaz, and Sihem Amer-Yahia. 2022. Guided Text-based Item Exploration. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 3410–3420. https://doi.org/10.1145/3511808.3557141

[15] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2778–2787.

[16] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Équille, Maximilian Fabricius, and Srividya Subramanian. 2021. Balancing Familiarity and Curiosity in Data Exploration with Deep Reinforcement Learning. In *aiDM '21: Fourth Workshop in Exploiting AI Techniques for Data Management, Virtual Event, China, 25 June, 2021*. 16–23.

[17] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Équille, Maximilian Fabricius, and Srividya Subramanian. 2021. DORA THE EXPLORER: Exploring Very Large Data With Interactive Deep Reinforcement Learning. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. 4769–4773.

[18] Aurélien Personnaz, Brit Youngmann, and Sihem Amer-Yahia. 2022. EDA4SUM: Guided Exploration of Data Summaries. *Proc. VLDB Endow.* 15, 12 (2022), 3590–3593.

[19] David A. Randell, Zhan Cui, and Anthony G. Cohn. 1992. A Spatial Logic Based on Regions and Connection. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*. 165–176.

[20] Mariia Seleznova, Behrooz Omidvar-Tehrani, Sihem Amer-Yahia, and Eric Simon. 2020. Guided exploration of user groups. *Proceedings of the VLDB Endowment (PVLDB)* 13, 9 (2020), 1469–1482.

[21] Shreya Shankar, J. D. Zamfirescu-Pereira, Björn Hartmann, Aditya G. Parameswaran, and Ian Arawjo. 2024. Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences. *CoRR* abs/2404.12272 (2024).

[22] Jiayang Song, Zhehua Zhou, Jiawei Liu, Chunrong Fang, Zhan Shu, and Lei Ma. 2023. Self-Refined Large Language Model as Automated Reward Function Designer for Deep Reinforcement Learning in Robotics. *CoRR* abs/2309.06687 (2023). https://doi.org/10.48550/ARXIV.2309.06687 arXiv:2309.06687

[23] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning*. MIT press.

[24] Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. *CoRR* abs/2306.03314 (2023). https://doi.org/10.48550/ARXIV.2306.03314 arXiv:2306.03314

[25] Immanuel Trummer and Christoph Koch. 2014. Approximation schemes for many-objective query optimization. In *SIGMOD*.

[26] Michele Tufano, Anisha Agarwal, Jinu Jang, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. 2024. AutoDev: Automated AI-Driven Development. *CoRR* abs/2403.08299 (2024). https://doi.org/10.48550/ARXIV.2403.08299 arXiv:2403.08299

[27] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. 2004. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI)*, Vol. 126.

[28] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[29] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. *CoRR* abs/2308.08155 (2023). https://doi.org/10.48550/ARXIV.2308.08155 arXiv:2308.08155

[30] Weiwen Xu, Deng Cai, Zhisong Zhang, Wai Lam, and Shuming Shi. 2024. Reasons to Reject? Aligning Language Models with Judgments. arXiv:2312.14591 [cs.CL] https://arxiv.org/abs/2312.14591

[31] Brit Youngmann, Sihem Amer-Yahia, and Aurélien Personnaz. 2022. Guided Exploration of Data Summaries. *Proc. VLDB Endow.* 15, 9 (2022), 1798–1807.

[32] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. 2009. It takes variety to make a world: diversification in recommender systems. In *EDBT*.

[33] Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. 2023. Large Language Models Are Semi-Parametric Reinforcement Learning Agents. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/f6b22ac37beb5da61efd4882082c9ecd-Abstract-Conference.html

[34] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. *CoRR* abs/2310.04406 (2023). https://doi.org/10.48550/ARXIV.2310.04406 arXiv:2310.04406