

Asking for Knowledge: Training RL Agents to Query External Knowledge Using Language

Iou-Jen Liu, Xingdi Yuan, Marc-Alexandre Côté, Pierre-Yves Oudeyer, Alexander G Schwing

▶ **To cite this version:**

Iou-Jen Liu, Xingdi Yuan, Marc-Alexandre Côté, Pierre-Yves Oudeyer, Alexander G Schwing. Asking for Knowledge: Training RL Agents to Query External Knowledge Using Language. ICML 2022 - 39th International Conference on Machine Learning, Jul 2022, Baltimore, United States. hal-03897379

HAL Id: hal-03897379
https://hal.science/hal-03897379

Submitted on 13 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Asking for Knowledge : Training RL Agents to Query External Knowledge Using Language

Iou-Jen Liu [*†1] **Xingdi Yuan** [*2] **Marc-Alexandre Côté** [*2] **Pierre-Yves Oudeyer** [†23] **Alexander G. Schwing** [1]

Abstract

To solve difficult tasks, humans ask questions to acquire knowledge from external sources. In contrast, classical reinforcement learning agents lack such an ability and often resort to exploratory behavior. This is exacerbated as few present-day environments support querying for knowledge. In order to study how agents can be taught to query external knowledge via language, we first introduce two new environments: the grid-world-based *Q-BabyAI* and the text-based *Q-TextWorld*. In addition to physical interactions, an agent can query an external knowledge source specialized for these environments to gather information. Second, we propose the 'Asking for Knowledge' (AFK) agent, which learns to generate language commands to query for meaningful knowledge that helps solve the tasks. AFK leverages a non-parametric memory, a pointer mechanism and an episodic exploration bonus to tackle (1) irrelevant information, (2) a large query language space, (3) delayed reward for making meaningful queries. Extensive experiments demonstrate that the AFK agent outperforms recent baselines on the challenging *Q-BabyAI* and *Q-TextWorld* environments. The code of the environments and agents are available at https://ioujenliu.github.io/AFK.

1. Introduction

To solve challenging tasks, humans query external knowledge sources, *i.e.*, we ask for help. We also constantly create knowledge sources (*e.g.*, manuals), as it is often more economical in the long term than users exploring via trial and error. Moreover, cognitive science research (Maratsos, 2007; Mills et al., 2010; Ronfard et al., 2018) showed that learning to ask questions and to interpret answers is key in a child's development of problem-solving skills. Consequently, we hypothesize that autonomous agents can address more complicated tasks if they can successfully learn to query external knowledge sources. For querying, it seems desirable to use some form of language. Not only does this allow to leverage existing knowledge sources built for humans, but also does it enable us to interpret the queries.

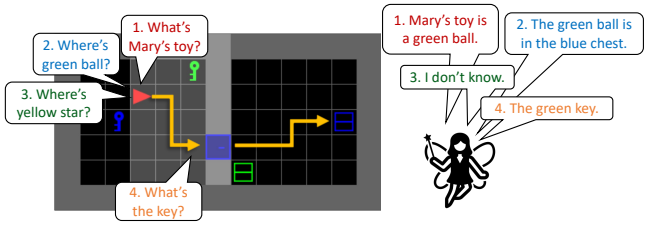


Figure 1. Proposed *Q-BabyAI*. Here the agent has to query the knowledge source to succeed.

However, the literature to teach agents to query external knowledge sources via language is scarce. Nguyen & Daume (2019) consider agents that can request help in visual navigation tasks. The agent can issue a 'help' signal, and expects the environment to provide an instruction leading it to a place towards the goal. Hence, agents learn when to query, but do not have control of what to query. Zhong et al. (2020) show that agents can better address novel tasks when a manual is available. However, the manual contains all relevant information and agents don't need to learn to query. Kovac et al. (2021) discuss the open challenge of building agents that learn social interaction skills mixing physical action and language. They show that state-of-the-art deep reinforcement learning systems cannot learn several kinds of social interaction skills. Instead of social skills, we focus on the open challenge of learning to ask for knowledge using language and propose an effective approach.

To deliberately study how agents can be taught to

[*]Equal contribution †Work partially done while visiting MSR [1]University of Illinois at Urbana-Champaign, IL, U.S.A. [2]Microsoft Research, Montréal, Canada [3]Inria, France. Correspondence to: Iou-Jen Liu <iliu3@illinois.edu>, Xingdi Yuan <eric.yuan@microsoft.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

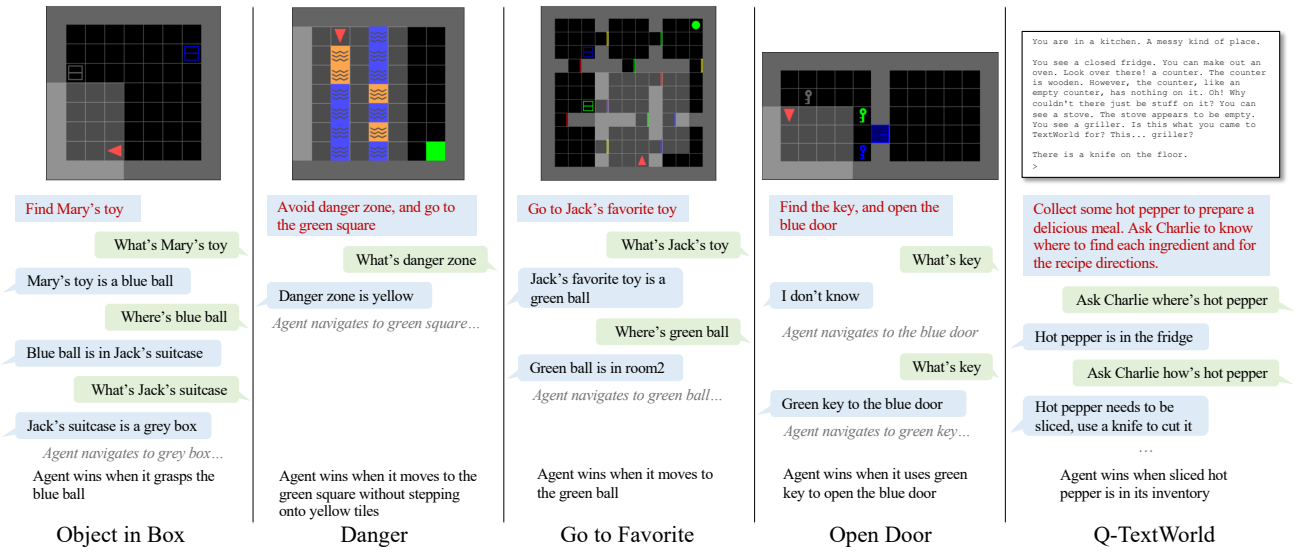


Figure 2. Querying interactions in *Q-BabyAI* and *Q-TextWorld*. We illustrate standard physical interactions as gray colored stage directions and highlight the questions (green) and oracle replies (blue) upon receiving the instruction (red).

query, we introduce two environments: the grid-world-based *Q-BabyAI*, illustrated in Fig. 1 and inspired by BabyAI (Chevalier-Boisvert et al., 2019), and the text-based *Q-TextWorld* inspired by TextWorld (Côté et al., 2018). In addition to physical interactions, an agent can use a query language to gather information related to a task. Importantly, in *Q-BabyAI* and *Q-TextWorld*, the knowledge source is designed to be task-agnostic, *i.e.*, it replies to all queries, even if irrelevant to the task at hand. This mimics many real-world knowledge sources, *e.g.*, search engines, which return results based on a user's query, regardless of relevance.

When training agents to query external knowledge via language, three main challenges arise: (1) The action space for generating a language query is large. Even with a template language, the action space grows combinatorially and large action spaces remain a challenge for reinforcement learning (Dulac-Arnold et al., 2016; Ammanabrolu & Hausknecht, 2020). (2) Irrelevant knowledge queried from a task-agnostic knowledge source can confuse agents. As a result, learning to ask meaningful questions is critical. This challenge is in line with the cognitive science finding (Mills et al., 2010) that children must learn to ask questions that result in answers with useful information. (3) Rewards for queries are often significantly delayed and sparse. Since the knowledge source provides specific information rather than a solution, agents have to also understand how to use the acquired information before receiving a significant reward. This mimics the discovery of Mills et al. (2010) that children must learn to use the received information.

To address the three challenges, we propose the 'asking for knowledge' (AFK) agent. The AFK agent is equipped with a pointer mechanism and a non-parametric memory, which we refer to as a 'notebook,' while using an episodic exploration strategy. The pointer mechanism addresses the challenge of a combinatorially growing action space by restricting the available actions based on the current context. The notebook keeps track of all the information related to the task at hand. The episodic exploration strategy deals with delayed and sparse rewards by issuing an exploration bonus when the agent makes novel and meaningful queries, inspired by information-seeking and epistemic curiosity observed in children (Engel, 2011; Gottlieb et al., 2013; Kidd & Hayden, 2015).

Comparing this AFK agent to recent baselines on *Q-BabyAI* and *Q-TextWorld*, we observe the AFK agent to ask more meaningful questions and to better leverage the acquired knowledge to solve the tasks.

2. Queryable Environments

We first discuss a reinforcement learning (RL) context where agents can query. We then introduce two new environments, *Q-BabyAI* and *Q-TextWorld*, each expanded from prior work (Chevalier-Boisvert et al., 2019; Côté et al., 2018).

2.1. Problem Setting

Reinforcement learning considers an agent interacting with an environment and collecting reward over discrete time. The environment is formalized by a partially observable Markov Decision Process (POMDP) (Sutton &

Barto, 2018). Formally, a POMDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{T}, \mathcal{O}, R, \gamma, H)$. $\mathcal{S}$ is the state space. $\mathcal{A}$ is the action space. $\mathcal{Z}$ is the observation space. At each time step t, the agent receives an observation $o_t \in \mathcal{Z}$ following the observation function $\mathcal{O} : \mathcal{S} \to \mathcal{Z}$ and selects an action $a_t \in \mathcal{A}$. The transition function $\mathcal{T}$ maps the action a_t and the current state s_t to a distribution over the next state s_{t+1}, *i.e.*, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$. The agent receives a real-valued reward r_t according to a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The agent's goal is to maximize the return $\sum_{t=0}^{H} \gamma^t r_t$, where γ is the discount factor and H is the horizon.

In a queryable environment, in addition to observations representing its surrounding, an agent also receives a response from the knowledge source upon issuing a query. Formally, the observation space $\mathcal{Z} = \mathcal{Z}_{\text{env}} \times \mathcal{Z}_q$ is composed of $\mathcal{Z}_{\text{env}}$ and $\mathcal{Z}_q$, representing the agent's surrounding and the reponse to a query, respectively.

Similarly, at each step, the agent's action space $\mathcal{A} = \mathcal{A}_{\text{phy}} \cup \mathcal{A}_q$ is composed of the physical action space $\mathcal{A}_{\text{phy}}$ supported by classical RL environments (*e.g.*, navigational actions, toggle, grasp) and the query action space $\mathcal{A}_q$.

Response Space $\mathcal{Z}_q$ and Query Action Space $\mathcal{A}_q$: As a controllable starting point for this research, we equip the environments with a queryable oracle knowledge source. Specifically, whenever receiving a sequence of tokens as a query, the oracle replies with a sequence of tokens. To consider the compositionality of language while reducing the burden of precise natural language generation, we define a template format for queries and responses. This design is also compatible with our plan of extending the knowledge source to more natural forms like databases.

A query is defined as a 3-tuple of `<func, adj, noun>`. In this 3-tuple, `func` is a function word selected from words like `where's`, `what's` and `how's`, which indicates the function of a query (*e.g.*, inquire about an object's location or affordances). The combination of an adjective (`adj`) and a `noun` enables to refer to a unique object within the environment.

Given a query, the oracle replies with a sequence of tokens. For this, the oracle has access to a set of "knowledge facts" associated with a particular instantiation of the environment. The knowledge facts are key-value pairs, where keys are the aforementioned 3-tuple of `<func, adj, noun>` and values are sequences of tokens. If a given query matches a key in the set of knowledge facts, the oracle will return the corresponding value. Otherwise, the oracle returns the message `I don't know`.

Crucially, the set of knowledge facts is much larger than necessary and irrelevant information is, by design, accessible to the agent. For instance, when tasked to find Mary's toy, information about Tim and Tim's toy is also available if queried. Gathering irrelevant information may lead to confusion and subsequent sub-optimal decisions. Moreover, some tasks require multi-hop information gathering (*e.g.*, **Object in Box**), in which the agent must ask follow-up questions to get all information needed to solve it.

Information Sufficiency: Practically, agents that can query have two main advantages. First, for environments containing sufficient information to be solved via exhaustive exploration, querying can provide a more natural and effective way to gather information (*e.g.*, reducing the policy length). Second, for environments that only provide partial information (*e.g.*, an agent must recognize and avoid danger tiles by trial-and-error, but danger tiles are randomly assigned per episode), only querying will lead to successful completion of the tasks.

To study both advantages, we augment BabyAI (Chevalier-Boisvert et al., 2019) and TextWorld (Côté et al., 2018) with a queryable knowledge source. We design tasks where the environment contains sufficient information, but we add knowledge facts which can help the agent to reduce exploration if used adequately. In addition, we design other tasks where agents can only succeed when they are able to query. We provide details next.

2.2. *Q-BabyAI*

We first introduce *Q-BabyAI*, an extension of the BabyAI environment (Chevalier-Boisvert et al., 2019). We devise four level 1 tasks, namely **Object in Box**, **Danger**, **Go to Favorite** and **Open Door**. In Fig. 2, we provide examples of agents querying the knowledge source for each of the level 1 tasks after receiving the goal instruction for that episode. The four tasks permit to study the two advantages mentioned above.

Specifically, both the **Object in Box** and **Danger** tasks can only be solved 100% of the time when querying is used to reveal the necessary knowledge — opening the wrong box or stepping on the danger tile terminates the game. In contrast, for the **Go to Favorite** and **Open Door** tasks, an agent can exhaust the environment to accomplish the goals. However, querying the knowledge source can greatly boost the agent's efficiency in both tasks. To prevent agents from memorizing solutions (*e.g.*, Mary's toy is in the red box), we randomly place objects and tiles in the environment, as well as shuffle the entity names and colors in every episode.

For the **Object in Box** and **Danger** tasks, we use a single-room setting to separate the difficulties of navigation and querying. In the **Go to Favorite** and **Open Door** tasks, we use a multi-room setting. It is worth noting that in the **Open Door** task, only querying at specific locations (*i.e.*, next to doors) can result in meaningful answers.

Having the four level 1 tasks defined, we increase the diffi-

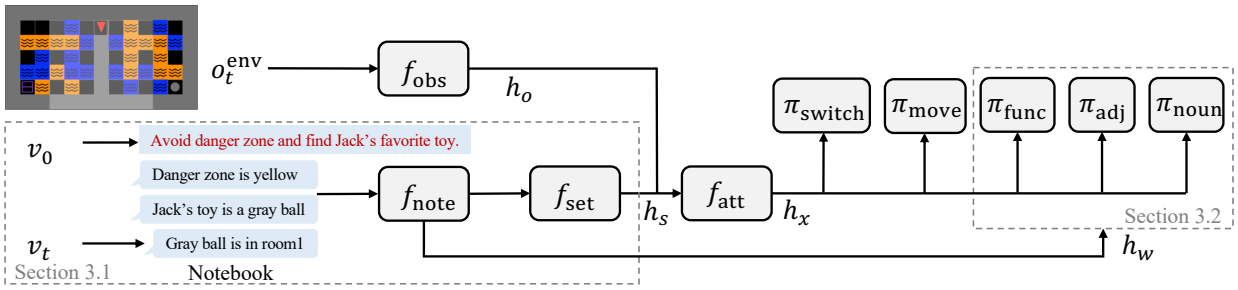


Figure 3. An overview of the AFK agent. An embedding of the notebook (h_s; see Sec. 3.1) and the environment (h_o) is combined (h_x) for use in five policy functions. The policies for query generation make use of notebook information (h_w) (see Sec. 3.2).

culty by composing them into more challenging higher-level tasks. For level k tasks, we combine k different tasks selected from the four level 1 tasks. As a result, we have six level 2 tasks, four level 3 tasks and one level 4 task. As an example, **Open Door + Object in Box** is a level 2 task where the instruction could be `Find the key to the door, and find Mary's toy`. To solve the task, an agent must figure out what is Mary's toy, where it is, and which key opens the locked door. We provide full details of the *Q-BabyAI* tasks including statistics in Appendix A.1.

2.3. *Q-TextWorld*

We also develop *Q-TextWorld*, augmenting the TextWorld environment (Côté et al., 2018) with a queryable knowledge source. Given a few configuration parameters, *Q-TextWorld* can generate new text-based games on the fly. Those interactive text environments are POMDPs with text-only observations and action space. Agents interact with the game by issuing short text phrases as actions, then get textual feedback. Text-based games provide a different view from their vision- or grid-based counterparts that can make things harder: 1) states are represented by highly abstracted signals requiring language understanding to interpret correctly; 2) the action space is combinatorially large due to compositionality of language; 3) levels of verbosity, *i.e.*, amount of irrelevant text, can potentially confuse an agent.

In this work, we adopt the cooking themed setting from prior work (Adhikari et al., 2020). An example is shown in Fig. 2. In all games, an agent must gather cooking ingredients, which are placed randomly in the room, either visible to the agent, or hidden inside some containers that need to be opened first. In a more difficult setting, each ingredient also needs to be cut in a specific way (*i.e.*, **`chopped`**, **`sliced`**, or **`diced`**). The agent must query the knowledge source to obtain that information and then act accordingly by issuing the right action while holding both the ingredient and a knife. We provide full details of the *Q-TextWorld* tasks including statistics in Appendix A.2.

Being consistent with the *Q-BabyAI* tasks, we study both advantages of having a querying behavior — improve efficiency and acquire necessary information. In games where cutting is not involved, agents can rely on exhaustive exploration to gather all portable objects and win. However, knowing what and where the required ingredients are can improve efficiency significantly. In contrast, in games where the ingredients need to be cut, an incorrect operation on the ingredient terminates the game. Hence, querying the recipe is the only way to perform above random.

3. Asking for Knowledge (AFK) Agent

In this section, we first present an overview of the 'Asking for Knowledge' (AFK) agent before we discuss details.

Overview: As illustrated in Fig. 3, the goal of the agent is to solve a task specified by an instruction. The language-based instruction (sequence of tokens) v_0 is provided at the start of each episode. At each time step t, the agent receives an environment observation $o_t^{\text{env}} \in \mathcal{Z}_{\text{env}}$. Moreover, if the agent issued a query at time step $t-1$, it also receives a language response $v_t \in \mathcal{Z}_q$ from the oracle, otherwise $v_t = \varnothing$.

To reduce the amount of noisy information (*i.e.*, v_t unrelated to the task at hand), we develop a non-parametric memory for gathered information, which we refer to as a 'notebook.' The notebook is a collection of sets where related information are being combined into a single set. The AFK agent only looks at the set that contains the task instruction v_0, which determines relevance to the task. Upon processing the notebook we obtain a representation h_s (details in Sec. 3.1).

We combine the environment observation o_t^{env} and the notebook representation h_s relevant for the task via an aggregator module (Perez et al., 2018; Vaswani et al., 2017). Given the output of the aggregator, $h_x \in \mathbb{R}^l$, where l is the encoding size, we use five heads to generate the physical actions and the language query actions. Specifically, we use a switch head $\pi_{\text{switch}}(\cdot|h_x) : \mathbb{R}^l \to \Delta(\{0,1\})$, a physical action head $\pi_{\text{phy}}(\cdot|h_x) : \mathbb{R}^l \to \Delta(\mathcal{A}_{\text{phy}})$, a function word head: $\pi_{\text{func}}(\cdot|h_x) : \mathbb{R}^l \to \Delta(V_{\text{func}})$, an adjec-

tive head $\pi_{\text{adj}}(\cdot|h_x) : \mathbb{R}^l \to \Delta(V_{\text{adj}})$, and a noun head $\pi_{\text{noun}}(\cdot|h_x) : \mathbb{R}^l \to \Delta(V_{\text{noun}})$. Here, $\Delta(X)$ represents a distribution with support X, $\mathcal{A}_{\text{phy}}$ is the physical action space, and $V_{\text{func}}, V_{\text{adj}}, V_{\text{noun}}$ are the function word, adjective, and noun vocabulary spaces.

The switch head decides whether the agent executes a physical action or issues a query. Conceptually, the agent first samples a value z according to $\pi_{\text{switch}}(\cdot|h_x)$. If $z = 0$, the agent will sample a physical action a_{phy} from $\pi_{\text{phy}}(\cdot|h_x)$ which is subsequently executed. In contrast, if $z = 1$, the agent will issue the query $[w_{\text{func}}, w_{\text{adj}}, w_{\text{noun}}]$ by sampling a function word w_{func}, an adjective w_{adj}, and a noun w_{noun} independently from $\pi_{\text{func}}(\cdot|h_x)$, $\pi_{\text{adj}}(\cdot|h_x)$, and $\pi_{\text{noun}}(\cdot|h_x)$. Note that the query action space $\mathcal{A}_q$ is large, which makes RL training challenging. We provide detailed statistics in Appendix A.

To alleviate issues due to the large action space, we adopt a pointer network (See et al., 2017) to generate queries. Specifically, the pointer network is restricted to 'point' to words occurring in the notebook. This ensures that the generated query uses words that are related to the already gathered information (details in Sec. 3.2).

In addition, to deal with delayed and sparse rewards, we propose an episodic exploration method which further incentivizes an agent to ask questions that are related to the task at hand (details in Sec. 3.3).

3.1. Notebook

In the following, we discuss the notebook's construction and describe the computation of the encoding h_s.

Notebook construction: Let F (for facts) denote the notebook, which is a non-parametric memory. Formally, F is a set of disjoint sets, *i.e.*, $F = \{A_i\}_{i=0}^{|F|-1}$ and $A_i \cap A_j = \varnothing, \forall i \neq j$. For each set A_i, each element $v \in A_i$ represents either a response from the oracle or the task instruction.

At the beginning of each episode, the notebook F is initialized with a singleton $A_0 = \{v_0\}$ that contains the task instruction v_0, *i.e.*, $F = \{\{v_0\}\}$. When an agent receives a new response $v_i \neq \varnothing$, we first find all sets that contain information related to v_i in the notebook. Formally, we construct an index set S that consists of the indices of related sets, *i.e.*, $S = \{j | \exists v \in A_j \text{ s.t. } \text{Sim}(v_i, v) \geq \alpha\}$, where $\text{Sim}(u, v) \in [0, 1]$ is a similarity function and $\alpha \in [0, 1]$ is a threshold. We study both the uni-gram and bi-gram similarity (Kondrak, 2005). If S is not empty, we combine all the related sets and the new response v_i to obtain a new set A_k. Formally, $A_k = \bigcup_{j \in S} A_j \cup \{v_i\}$, where $k = \min_{j \in S} j$. Then, all sets $\{A_j\}_{j \in S}$ are replaced with the new set A_k. If the index set S is empty, we add $A_k = \{v_i\}$ to F, where k is the next available index. Importantly, note that the task instruction v_0 is always part of the set A_0.

Notebook encoding: To discard noisy information coming from responses unrelated to the task at hand, the AFK agent only considers the set A_0 which contains the task instruction v_0. We use a recurrent neural network f_{note} (Cho et al., 2014) to encode each 'note' in A_0, *i.e.*, for each $v_i \in A_0$ ($i \in \{1, \ldots, |A_0|\}$), we have $h_i = f_{\text{note}}(v_i) \in \mathbb{R}^{|v_i| \times l}$, where $|v_i|$ is the number of words in v_i and l is the hidden dimension. To further encode the instruction related notes (*i.e.*, A_0) as a whole, we use a Deep Set model f_{set} (Zaheer et al., 2017), *i.e.*, $h_s = f_{\text{set}}([h_1, \ldots, h_{|A_0|}]) \in \mathbb{R}^l$, where h_s is the resulting encoding. In addition, the input observation o^{env} is encoded via a neural network f_{obs}, *i.e.*, $h_o = f_{\text{obs}}(o^{\text{env}}) \in \mathbb{R}^l$, where h_o is the resulting observation encoding. An aggregator module is used to combine h_o and h_s, *i.e.*, $h_x = f_{\text{att}}(h_o, h_s) \in \mathbb{R}^l$.

3.2. Pointer Mechanism for Language Generation

To address the challenge of a combinatorially growing action space, we develop a pointer mechanism for the policies π_{adj} and π_{noun}. Concretely, the pointer mechanism restricts the AFK agent queries to use only the words appearing in the set A_0.

We achieve this by first applying a mask before computing the policy distributions π_{adj} and π_{noun}, *i.e.*, π_{adj} and π_{noun} only have non-zero probability for adjectives and nouns in the instruction related set of notes A_0. We use the generation process of the noun as an example. Let m_{noun} denote the number of nouns in A_0, and let $h_w \in \mathbb{R}^{m_{\text{noun}} \times l}$ denote the word encodings of all nouns in A_0. Using attention queries $q \in \mathbb{R}^l$ and keys $k \in \mathbb{R}^{m_{\text{noun}} \times l}$ such that

$$q = h_x \cdot W_q, \quad \text{and} \quad k = h_w \cdot W_k, \tag{1}$$

with learnable parameters $W_q, W_k \in \mathbb{R}^{l \times l}$, we compute the attention e_{noun} over all nouns in A_0 as

$$e_{\text{noun}} = \text{softmax}(q \cdot k^T) \in \mathbb{R}^{m_{\text{noun}}}. \tag{2}$$

A distribution over the noun vocabulary, *i.e.*, V_{noun}, is then constructed from e_{noun}. Specifically, for each word $w \in V_{\text{noun}}$, we have $\pi_{\text{noun}}(w) = \sum_{i=1}^{m_{\text{noun}}} e^i_{\text{noun}} \text{I}[d(i) = w]$, where $d(i)$ maps the index i to the corresponding word in A_0 and e^i_{noun} represents the i-th element of e_{noun}. I is the indicator function. The pointer mechanism for π_{adj} is constructed similarly. We defer details to Appendix C.

3.3. Episodic Exploration

To deal with delayed and sparse rewards, inspired by Savinov et al. (2019), we develop an episodic exploration mechanism to encourage the agent to ask questions related to the task at hand.

	Tasks	No Query	Query Baseline	AFK (Ours)
Lv. 1	♣	50.5±2.0	49.8±1.2	**100.0±0.0**
	♠	68.3±2.4	73.8±1.2	**100.0±0.0**
	♦	98.9±0.8	99.3±0.3	**100.0±0.0**
	♥	99.7±0.3	85.3±22.3	**100.0±0.0**
Lv. 2	♣♠	0.0±0.0	0.0±0.0	**90.3±1.8**
	♣♦	0.1±0.1	0.6±0.5	**94.3±2.3**
	♣♥	0.0±0.0	0.0±0.0	**99.0±0.4**
	♠♦	0.4±0.1	0.2±0.2	**100.0±0.0**
	♠♥	0.0±0.0	0.0±0.0	0.0±0.0
	♦♥	84.1±0.3	94.0±3.3	**98.7±0.2**
Lv. 3	♣♠♦	0.0±0.0	0.0±0.0	0.15±0.2
	♣♠♥	0.0±0.0	0.0±0.0	0.0±0.0
	♣♦♥	0.0±0.0	0.0±0.0	2.1±0.8
	♠♦♥	4.3±1.0	4.4±0.8	4.8±0.9
Lv. 4	♣♠♦♥	0.0±0.0	0.0±0.0	0.0±0.1

Table 1. Success rate (%) on *Q-BabyAI*. ♣: **Object in Box**, ♠: **Danger**, ♦: **Go to Favorite**, ♥: **Open Door**.

Tasks	No Query	Query Baseline	AFK (Ours)
♦	30.2±1.5	26.7±1.5	**16.8±6.7**
♥	26.2±0.9	36.8±1.0	**20.6±0.2**

Table 2. Number of steps required to solve a task. ♦: **Go to Favorite**, ♥: **Open Door**.

At each time step, the agent receives reward $r = r^{\text{env}} + b$, where r^{env} is the external reward and b is the bonus reward. A positive bonus reward b is obtained whenever a query's response $v_i \neq \varnothing$ expands the agent's knowledge about the task, *i.e.*, A_0. The reward is only given for new information. Formally,

$$b = \beta(\mathbb{I}[(v_i \in A_0) \wedge (v_i \notin A_0')]), \tag{3}$$

where v_i denotes a new response returned by the oracle, and A_0' denotes the set from the previous game step containing the task instruction v_0. $\beta > 0$ is a scaling factor and I is the indicator function.

4. Experimental Results

In this section, we present the experimental setup, evaluation protocol, and our results on *Q-BabyAI* and *Q-TextWorld*.

Experimental Setup: We adopt the publicly available BabyAI and TextWorld code released by the authors[1,2] as our non-query baseline system, denoted as **No Query**. We consider a vanilla query agent (Kovac et al., 2021) (**Query Baseline**), in which query heads are added to the baseline agent to generate language queries. We refer to the proposed agent via **AFK**, which is the agent with 1) notebook, 2) pointer mechanism, and 3) episodic exploration.

[1]BabyAI: github:mila-iqia/babyai

[2]TextWorld: github:xingdi-eric-yuan/qait_public

Task	No Query	Query Baseline	AFK (Ours)
Take 1	75.1±4.1	73.5±5.8	**85.1±2.9**
Take 2	24.0±6.6	13.7±8.5	**61.9±6.5**
Take 1 Cut	24.6±1.0	22.9±3.6	**43.5±15.9**
Take 2 Cut	0.0±0.0	0.0±0.0	0.0±0.0

Table 3. Success rate (%) on *Q-TextWorld*.

Task	AFK w/o Notebook	AFK w/o Pointer Mechanism	AFK w/o Episodic Exploration	AFK (Ours)
♣	50.0±0.8	49.4±0.7	49.8±0.7	**100.0±0.0**
♠	99.1±0.2	100.0±0.0	93.8±0.7	**100.0±0.0**
♦	99.2±0.4	99.7±0.2	99.3±0.2	**100.0±0.0**
♥	85.1±1.0	100.0±0.0	77.8±0.7	**100.0±0.0**
♣♥	48.5±1.9	90.5±1.4	50.0±1.8	**99.0±0.4**
Mean	76.4	87.9	74.1	99.8

Table 4. Ablation Study. Success rate (%) on *Q-BabyAI*. ♣: **Object in Box**, ♠: **Danger**, ♦: **Go to Favorite**, ♥: **Open Door**.

We follow the original training protocols used in BabyAI and TextWorld. Specifically, we train all agents in *Q-BabyAI* environments with proximal policy optimization (PPO) (Schulman et al., 2017) for 20M - 50M environment steps, depending on the tasks' difficulty. For *Q-TextWorld* agents, we use the Deep Q-Network (Mnih et al., 2013; Hessel et al., 2018) and the agents are trained for 500K episodes. We provide implementation details in Appendix C.

Evaluation Protocol: In *Q-BabyAI*, the policy is evaluated in an independent evaluation environment every 50 model updates and each evaluation consists of 500 evaluation episodes. To ensure a fair and rigorous evaluation, we follow the evaluation protocols suggested by Henderson et al. (2017); Colas et al. (2018) and report the 'final metric'. The final metric is the average evaluation success rate of the last ten models in the training process, *i.e.*, average success rate of the last 5000 evaluation episodes. In *Q-TextWorld*, we report the final running average training scores with a window size of 1000. Note, in each episode, entities are randomly spawned preventing agents from memorizing training games. All experiments are repeated five times with different random seeds.

***Q-BabyAI* Results:** We first compare our AFK agent with baselines on all level 1 and level 2 tasks of *Q-BabyAI*. The final metrics and standard deviation of average evaluation success rate are reported in Tab. 1. As shown in Tab. 1, for level 1 and level 2 tasks, the AFK agent achieves significantly higher success rates than the baselines, particularly in **Object in Box** (♣) and **Danger** (♠) where information

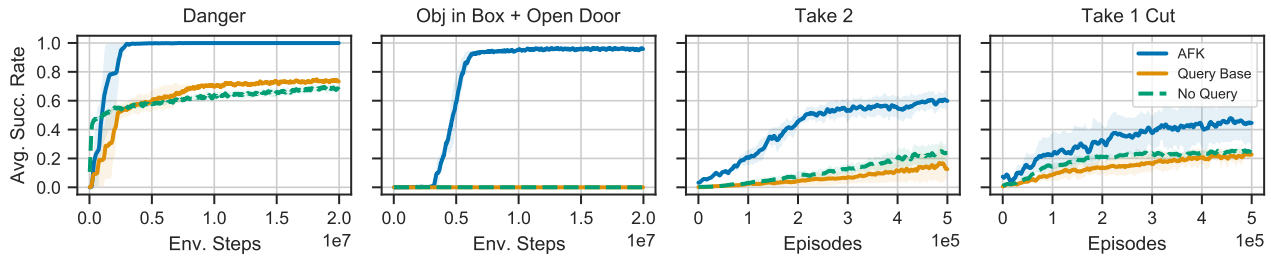


Figure 4. Training curves of AFK, non-query baseline, query baseline on *Q-BabyAI* (left) and *Q-TextWorld* (right).

Target Task	Source Tasks	Succ. (%)	Eps. Len.
♣♠	♠♦+♦♣	22.1±0.7	32.0±0.3
♠♦	♠♣+♦♣	35.6±2.1	71.6±1.1

Table 5. Zero-shot generalization study of AFK.

Task	$\lvert Q_t \rvert$	Precision	Recall	F1
♣♥	5	0.804	0.823	0.812
♦♥	4	0.771	0.560	0.601
♠♦	3	0.989	0.989	0.989

Table 6. Query quality of AFK. ♣: **Object in Box**, ♠: **Danger**, ♦: **Go to Favorite**, ♥: **Open Door**.

has to be queried. This demonstrates that the AFK agent asks more meaningful questions and successfully leverages oracle replies to solve tasks. We provide extra analysis to support this in a later subsection. In addition, we observe that in tasks where the instruction provides sufficient information, *e.g.*, **Go to Favorite** (♦) and **Open Door** (♥), all agents are able to solve the tasks. However, as shown in Tab. 2, an AFK agent needs fewer steps to solve the tasks. This suggests that meaningful queries can result in better efficiency. Training curves are shown in Fig. 4. See Appendix E for training curves and results of all experiments.

To show the limitation of the proposed approach, we run experiments on the very challenging level 3 and level 4 tasks of *Q-BabyAI*. As shown in Tab. 1, AFK as well as all baselines fail to solve the level 3 and level 4 tasks due to the tasks' high complexity and very sparse rewards. This shows that training RL agents to query in language is still a very challenging and open problem which needs more attention from our community.

***Q-TextWorld* Results:** We compare AFK with the baseline agents on *Q-TextWorld* in Tab. 3. Specifically, we conduct experiments on four settings with gradually increasing difficulty. Here, 'Take k' denotes that an agent needs to collect k food ingredients, which may spawn in containers and are hence invisible to the agent before the container is opened. 'Cut' indicates that the collected food ingredients need to be cut in specific ways, for which the recipe needs to be queried. As shown in Tab. 3, AFK significantly outperform the baselines on three of the tasks. Analogously to *Q-BabyAI* experiments, the No Query agent sometimes outperforms the Query Baseline agent. We believe this is caused by 1) the larger action space of the Query Baseline compared to No Query, and 2) a missing mechanism helping the agent to benefit from queries — together they reduce the Query Baseline's chance to experience meaningful trajectories. We observe that none of the agents can get non-zero scores on the Take 2 Cut task. We investigate the agents' training reward curves (Fig. 10, Appendix E): while the baselines get 0 reward, AFK actually learns to obtain higher reward. We suspect that due to the richer entity presence in *Q-TextWorld*, and the resulting larger number of valid questions (connected to A_0), AFK may exploit the exploration bonus and ask more questions than necessary. This suggests that better reward assignment methods are needed for agents to perform in more complex environments.

Ablation Study: We perform an ablation study to examine the effectiveness of the proposed 1) notebook, 2) pointer mechanism, and 3) episodic exploration. For this, we use various level 1 and level 2 *Q-BabyAI* tasks. The results are reported in Tab. 4. As shown in Tab. 4, removing the notebook, the pointer mechanism, or the episodic exploration results in the success rate dropping by 22.9%, 11.4%, and 25.2% on average. This demonstrates that all three proposed components are essential for an AFK agent to successfully generate meaningful queries and solve tasks.

Generalization: To assess an AFK agent's capability of making meaningful queries and solving different, novel, unseen tasks, we perform a generalization study. Specifically, we train AFK agents on a set of level 2 source tasks. Then the trained AFK agent is tested on an unseen level 2 target task (new combination of sub-tasks used in training). The results are summarized in Tab. 5. As shown in Tab. 5, upon training on source tasks, an AFK agent achieves 22.1% and 35.6% success rate on the level 2 target tasks '**Object in Box + Danger**'(♣♠) and '**Danger + Go to Favorite**'(♠♦), which the agent has never seen during training. In contrast, the Query Baseline only achieves a 0.0% and 0.2% success rate on the target tasks after training 20M steps directly on

the target tasks (Tab. 1).

Query Quality: To gain more insights, we study the quality of the queries issued by an agent. Each episode of our tasks is associated with a set of queries Q_t which are useful for solving the task. If an agent issues a query $q \in Q_t$, the query is considered 'good.' We refer to the number of good queries (not counting duplicates) and total number of queries (counting duplicates) generated by the agent in one episode as n_g and n_{tot}. We report the average precision, recall, and F1 score (Sasaki, 2007) of the generated queries over 200 evaluation episodes. Specifically, precision = $\frac{n_g}{n_{\text{tot}}}$, recall = $\frac{n_g}{|Q_t|}$, and F1 score is the harmonic mean of precision and recall. As shown in Tab. 6, the AFK agent achieves high F1 scores across various tasks. In contrast, the Query Baseline converges to a policy that does not issue any query and thus has zero precision and recall in all tasks. This demonstrates AFK's ability to learn to ask relevant questions.

5. Related Work

Information Seeking Agents: In recent years a host of works discussed building of information seeking agents. Nguyen & Daume (2019) propose to leverage an oracle in 3D navigation environments. The oracle is activated in response to a special signal from the agent and provides a language instruction describing a subtask the agent could follow. Kovac et al. (2021) design grid-world tasks similar to ours, but focus on the social interaction perspective. For instance, some agents are required to emulate their social peers' behavior to successfully communicate with them. Potash & Suleman (2019) propose a game setting which requires agents to ask sequences of questions efficiently to guess the target sentence from a set of candidates. Yuan (2021); Nakano et al. (2021) propose agents that can generate sequences of executable commands (*e.g.*, Ctrl+F a token) to navigate through partially observable text environments for information gathering. The line of research on curiosity-driven exploration and intrinsic motivation shares the same overall goal to seek information (Oudeyer et al., 2007; Oudeyer & Kaplan, 2007). A subset of them, count-based exploration methods, count the visit of observations or states and encourage agents to gather more information from rarely experienced states (Bellemare et al., 2016; Ostrovski et al., 2017; Savinov et al., 2019; Liu et al., 2021). Our work also loosely relates to the active learning paradigm, where a system selects training examples wisely so that it achieves better model performance, while also consuming fewer training examples (Cohn et al., 1994; Bachman et al., 2017; Fang et al., 2017). Different from existing work, we aim to study explicit querying behavior using language. We design tasks where querying behavior can either greatly improve efficiency or is needed to succeed.

In a concurrent work, Nguyen et al. (2022) propose a framework tailored to 3D navigation environments: agents can query an oracle to obtain useful information (e.g., current state, current goal and subgoal). They show that navigation agents can take advantage of an assistance-requesting policy and improve navigation in unseen environments.

Reinforcement Learning with External Knowledge:

Training reinforcement learning agents which use external knowledge sources also received attention recently (He et al., 2017; Bougie & Ichise, 2017; Kimura et al., 2021; Argerich et al., 2020; Zhong et al., 2020). Various forms of external knowledge sources are considered. He et al. (2017) consider a set of documents as external knowledge source. An agent needs to learn to read the documents to solve a task. Bougie & Ichise (2017) consider environment information obtained by an object detector as external knowledge. They show that the additional information form the detector enables agents to learn faster. Kimura et al. (2021) consider a set of detailed instructions as knowledge source. They propose an architecture to aggregate the given external knowledge with the RL model. The aforementioned works assume the external knowledge is given and the agent doesn't need to learn to query. In contrast, we consider a task-agnostic interactive knowledge source. In our *Q-BabyAI* and *Q-TextWorld* environments, an agent must learn to actively execute meaningful queries in language to solve a task.

Question Generation and Information Retrieval: Question generation is a thriving direction at the intersection of multiple areas like natural language processing and information retrieval. In the machine reading comprehension literature, Du et al. (2017); Yuan et al. (2017); Jain et al. (2018) propose to reverse question answering: given a document and a phrase, a model is required to generate a question. The question can be answered by the phrase using the document as context. In later work, Scialom & Staiano (2020) define curiosity-driven question generation. Query reformulation is a technique which aims to obtain better answers from the knowledge source (*e.g.*, a search engine) by training agents to modify questions (Nogueira & Cho, 2017; Buck et al., 2018). Another loosely related area is multi-hop retrieval (Das et al., 2018; Xiong et al., 2021; Feldman & El-Yaniv, 2019), where a large scale supporting knowledge source is involved and systems must gather information in a sequential manner. Inspired by these works, we leverage properties of language such as compositionality, to help form a powerful query representation that is manageable by RL training.

6. Limitations and Future Work

In this section, we conclude by discussing limitations of this work and future directions.

Environments: As an initial attempt to study agents that learn to query knowledge sources with language, we settled on oracle-based knowledge sources. This ensures better experimental controllability and reproducibility. However, it can be improved in multiple directions.

1. Beyond the use of hand-crafted key-value pairs as the knowledge source, a set of more realistic knowledge sources can be considered. For instance, databases can be queried using similar template language (Zhong et al., 2017); an information retrieval system or a pre-trained question answering system can be used to extract knowledge from large scale language data (Lewis et al., 2021; Borgeaud et al., 2021); a search engine is naturally queryable (Nakano et al., 2021); pre-trained language models can be queried via prompt engineering (Huang et al., 2022); humans can also be a great knowledge source (Kovac et al., 2021).

2. The query grammar can be extended to be more natural and informative (*e.g.*, `Where's Mary's toy and where can I find it?`).

3. We plan to include tasks that require non-linear reasoning. This will further decrease agents' incentive to memorize an optimal trajectory, and presumably increase generalizability.

Agent design: For agents, future directions include:

1. When the state space is large (*e.g.*, in *Q-TextWorld*), agents sometimes keep on querying different question to exploit the exploration bonus. This demands a better reward assignment strategy, since agents performing in more complex environments may encounter this issue too.

2. It is worth exploring other structured knowledge representations (Ammanabrolu & Hausknecht, 2020) and parametric memories (Weston et al., 2015; Munkhdalai et al., 2019) beyond the notebook we used.

3. Asking questions essentially serves to reduce entropy. One could further use exploration strategies that maximize information gain (Houthooft et al., 2016).

Overall, we are excited by the challenges and opportunities posed by agents that are able to learn to query external knowledge while acting in their environments. We strive to call attention from researchers for the development of agents capable of querying external knowledge sources — we believe this is a strong and natural skill. We make an initial effort towards this goal, which hopefully can be proven to be valuable and helpful to the community.

7. Acknowledgement

This work is supported in part by Microsoft Research, the National Science Foundation under Grants No. 1718221, 2008387, 2045586, 2106825, MRI #1725729, NIFA award 2020-67021-32799, and AWS Research Awards.

References

Adhikari, A., Yuan, X., Côté, M.-A., Zelinka, M., Rondeau, M.-A., Laroche, R., Poupart, P., Tang, J., Trischler, A., and Hamilton, W. Learning dynamic belief graphs to generalize on text-based games. In *Proc. NeurIPS*, 2020.

Ammanabrolu, P. and Hausknecht, M. Graph constrained reinforcement learning for natural language action spaces. In *ICLR*, 2020.

Argerich, M. F., Furst, J., and Cheng, B. Tutor4rl: Guiding reinforcement learning with external knowledge. In *arXiv.*, 2020.

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. In *arXiv.*, 2016.

Bachman, P., Sordoni, A., and Trischler, A. Learning algorithms for active learning. In *Proc. ICML*, 2017.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying Count-Based Exploration and Intrinsic Motivation. In *Proc. NeurIPS*, 2016.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Driessche, G. v. d., Lespiau, J.-B., Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. In *arXiv.*, 2021.

Bougie, N. and Ichise, R. Deep reinforcement learning boosted by external knowledge. In *arXiv.*, 2017.

Buck, C., Bulian, J., Ciaramita, M., Gajewski, W., Gesmundo, A., Houlsby, N., and Wang, W. Ask the right questions: Active question reformulation with reinforcement learning. In *ICLR*, 2018.

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. In *ICLR*, 2019.

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 *Proc. EMNLP*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *arXiv.*, 2014.

Cohn, D., Atlas, L., and Ladner, R. Improving generalization with active learning. *Machine learning*, 1994.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *Proc. ICML*, 2018.

Côté, M.-A., Ákos Kádár, Yuan, X. E., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., Asri, L. E., Adada, M., Tay, W., and Trischler, A. Textworld: A learning environment for text-based games. In *Computer Games Workshop at ICML/IJCAI*, 2018.

Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., and McCallum, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *Proc. ICLR*, 2018.

Du, X., Shao, J., and Cardie, C. Learning to ask: Neural question generation for reading comprehension. In *Proc. ACL*, 2017.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. In *Proc. ICML*, 2016.

Engel, S. Children's need to know: Curiosity in schools. *Harvard educational review*, 2011.

Fang, M., Li, Y., and Cohn, T. Learning how to active learn: A deep reinforcement learning approach. In *Proc. EMNLP*, 2017.

Feldman, Y. and El-Yaniv, R. Multi-hop paragraph retrieval for open-domain question answering. In *Proc. ACL*, 2019.

Gottlieb, J., Oudeyer, P.-Y., Lopes, M., and Baranes, A. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*, 2013.

He, J., Ostendorf, M., and He, X. Reinforcement learning with external knowledge and two-stage q-functions for predicting popular reddit threads. In *arXiv.*, 2017.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Proc. AAAI*, 2017.

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 *Proc. AAAI*, 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997.

Houthooft, R., Chen, X., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Proc. NeurIPS*, 2016.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *arXiv.*, 2022.

Jain, U., Lazebnik, S., and Schwing, A. G. Two can play this Game: Visual Dialog with Discriminative Question Generation and Answering. In *Proc. CVPR*, 2018.

Kidd, C. and Hayden, B. Y. The psychology and neuroscience of curiosity. *Neuron*, 2015.

Kimura, D., Chaudhury, S., Wachi, A., Kohita, R., Munawar, A., Tatsubori, M., and Gray, A. Reinforcement learning with external knowledge by using logical neural networks. In *arXiv.*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

Kondrak, G. N-gram similarity and distance. In *SPIRE*, 2005.

Kovac, G., Portelas, R., Hofmann, K., and Oudeyer, P.-Y. Socialai: Benchmarking socio-cognitive abilities in deep reinforcement learning agents. In *arXiv.*, 2021.

Lewis, P., Stenetorp, P., and Riedel, S. Question and answer test-train overlap in open-domain question answering datasets. In *Proc. EACL*, 2021.

Liu, I.-J., Jain, U., Yeh, R., and Schwing, A. G. Cooperative Exploration for Multi-Agent Deep Reinforcement Learning. In *Proc. ICML*, 2021.

Maratsos, M. P. Children's questions: A mechanism for cognitive development: Commentary. *Monographs of the Society for Research in Child Development*, 2007.

Mills, C. M., Legare, C. H., Bills, M., and Mejias, C. Preschoolers use questions as a tool to acquire knowledge from different sources. *Journal of Cognition and Development*, 2010.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. In *arXiv.*, 2013.

Munkhdalai, T., Sordoni, A., wang, T., and Trischler, A. Metalearned neural memory. In *Proc. NeurIPS*, 2019.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. Webgpt:

Browser-assisted question-answering with human feedback. In *arXiv.*, 2021.

Nguyen, K. and Daume, H. Help, anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning. In *EMNLP*, 2019.

Nguyen, K., Bisk, Y., and III, H. D. Learning When and What to Ask: a Hierarchical Reinforcement Learning Framework. In *Proc. ICML*, 2022.

Nogueira, R. and Cho, K. Task-oriented query reformulation with reinforcement learning. In *Proc. EMNLP*, 2017.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. Count-based exploration with neural density models. In *Proc. ICML*, 2017.

Oudeyer, P.-Y. and Kaplan, F. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 2007.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 2007.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Potash, P. and Suleman, K. Playing log (n)-questions over sentences. *arXiv preprint arXiv:1908.04660*, 2019.

Ronfard, S., Zambrana, I. M., Hermansen, T. K., and Kelemen, D. Question-asking in childhood: A review of the literature and a framework for understanding its development. *Developmental Review*, 49:101–120, 2018.

Sasaki, Y. The truth of the f-measure. In *arXiv.*, 2007.

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. In *Proc. ICLR*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. In *arxiv*, 2017.

Scialom, T. and Staiano, J. Ask to learn: A study on curiosity-driven question generation. In *Proc. COLING*, 2020.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. In *ACL*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proc. NeurIPS*, 2017.

Weston, J., , Chopra, S., , and Bordes, A. Memory networks. In *Proc. ICLR*, 2015.

Xiong, W., Li, X., Iyer, S., Du, J., Lewis, P., Wang, W. Y., Mehdad, Y., Yih, S., Riedel, S., Kiela, D., and Oguz, B. Answering complex open-domain questions with multi-hop dense retrieval. In *Proc. ICLR*, 2021.

Yuan, X. Interactive machine comprehension with dynamic knowledge graphs. In *EMNLP*, 2021.

Yuan, X., Wang, T., Gulcehre, C., Sordoni, A., Bachman, P., Subramanian, S., Zhang, S., and Trischler, A. Machine comprehension by text-to-text neural question generation. In *arXiv.*, 2017.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Proc. NeurIPS*, 2017.

Zhong, V., Xiong, C., and Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. In *arXiv.*, 2017.

Zhong, V., Rocktäschel, T., and Grefenstette, E. Rtfm: Generalising to novel environment dynamics via reading. In *ICLR*, 2020.

Appendix: Asking for Knowledge : Training RL Agents to Query External Knowledge Using Language

The appendix is structured as follows:

1. In Sec. A, we provide the details of each task in *Q-BabyAI* and *Q-TextWorld*.
2. In Sec. B, we provide the model details of our AFK agent.
3. In Sec. C, we provide the implementation and training details for the AFK agent.
4. In Sec. D, we provide additional experimental results on *Q-BabyAI*.
5. In Sec. E, we provide training curves for all experiments on *Q-BabyAI* and *Q-TextWorld*.

The Python code of *Q-BabyAI*, *Q-TextWorld*, the AFK agent and all baselines are available at `https://ioujenliu.github.io/AFK`.

A. Environment and Task Details

A.1. *Q-BabyAI*

General information	
Word vocabulary size	62
Function word vocabulary size ($\lvert V_{\text{func}} \rvert$)	2
Adjective vocabulary size ($\lvert V_{\text{adj}} \rvert$)	22
Noun vocabulary size ($\lvert V_{\text{noun}} \rvert$)	24
# of Physical action	7
Visual range	7×7
# of object colors	6
# of actionable object types	4
# of names	2
# of danger zone colors	2

Table 7. Statistics of the *Q-BabyAI* environment.

The general statistics of *Q-BabyAI* tasks are summarized in Tab. 7. The statistics for each individual *Q-BabyAI* task are summarized in Tab. 8, where $\lvert Q_t \rvert$ represents the number of 'good queries' an agent should make to solve a task efficiently. 'Early Terminate' indicates that an episode will be terminated if the agent makes a mistake, *e.g.*, stepping on a danger zone or opening the wrong box. In addition, we present the details of the four basic *Q-BabyAI* tasks in the following.

Object in Box ♣: There are two suitcases in the environment. Each suitcase contains one toy. The instruction is **`find <name>'s toy`**, where **`<name>`** is sampled from a set of names at the start of each episode. However, the agent doesn't know what is the referred toy. Neither does it know the content of each suitcase. The episode terminates when a suitcase is opened by the agent. Therefore, the agent needs to ask multiple question to figure out what the desired toy is and which suitcase to open. If the opened suitcase contains the desired toy, the agent receives a positive reward. Otherwise, it doesn't receive any reward.

Danger ♠: There are different colors of tiles in the environment. One of the colors represents the danger zone. The episode terminates if the agent steps on a danger zone. The instruction is **`avoid danger zone, and go to the green target square`**. However, the agent doesn't know what color represents the danger zone. Therefore, to safely reach the target square and receive rewards, the agent must ask the oracle for information on the danger zone. Importantly, the color of the danger zone differs from episode to episode.

Go to Favorite ♦: There are nine rooms in the environment. The instruction is **`Go to <name>'s favorite toy`**, where **`<name>`** and **`<name>'s favorite toy`** are sampled from a set of names and a set of toys at the start of each episode. There are irrelevant objects scattered around the environment. To solve the task efficiently, the agent should issue queries to figure out what and where is the referred toy. Note, if the agent doesn't ask any question, it can still solve the task, but in a much less efficient manner, *i.e.*, by exhaustively searching all rooms for the referred toy. The agent receives positive reward when it goes to the referred toy.

Open Door ♥: There are three keys and one door in the environment. One of the three keys could open the door. The agent needs to find the right key and open the door to complete the task and receive a positive reward. The instruction is **`Find the key to the door`**. Note, the agent could still complete the task without asking any question, *i.e.*, by exhaustively trying all keys.

A.2. *Q-TextWorld*

For all games, the objective is to find cooking ingredients which are randomly hidden throughout the kitchen. Once found, those ingredients may require some processing depending on the task difficulty. Once all required ingredients are in the player's inventory and processed the right way, the game terminates with a reward of 1. We provide statistics of the *Q-TextWorld* environment in Tab. 9 and a transcript of a game can be seen in Fig. 5.

Take [1/2]: In this task, the player has to find 1 or 2 ingredients mentioned in the instruction. Ingredients are either

	Tasks	$\lvert Q_t \rvert$	# of rooms	room size	Early Terminate
Lv. 1	♣	3	1	9×9	True
	♠	1	1	7×7	True
	♦	2	9	5×5	False
	♥	1	2	7×7	False
Lv. 2	♣♠	5	2	7×7	True
	♣♦	4	9	5×5	True
	♣♥	5	2	7×7	True
	♠♦	3	2	7×7	True
	♠♥	2	2	7×7	True
	♦♥	4	9	5×5	False
Lv. 3	♣♠♦	5	2	7×7	True
	♣♠♥	6	3	7×7	True
	♣♦♥	5	9	5×5	True
	♠♦♥	4	3	7×7	True
Lv. 4	♣♠♦♥	7	9	7×7	True

Table 8. Statistics of each task in *Q-BabyAI*. ♣: **Object in Box**, ♠: **Danger**, ♦: **Go to Favorite**, ♥: **Open Door**.

Instruction: You find yourself at friend's house and you are both hungry! Collect some parsley to prepare a delicious meal. Ask Charlie to know where to find each ingredient and for the recipe directions.

Description: You are in a kitchen. A messy kind of place.

You see a closed fridge. You can make out an oven. Look over there! a counter. The counter is wooden. However, the counter, like an empty counter, has nothing on it. Oh! Why couldn't there just be stuff on it? You can see a stove. The stove appears to be empty. You see a griller. Is this what you came to TextWorld for? This... griller?

There is a cookbook on the floor.

Inventory: You are carrying: a knife.

Notebook: {"You find yourself . . . for the recipe directions."}

Action 1: **ask Charlie where's the parsley**

Feedback: The parsley is in the oven.

Notebook: {"You find yourself . . . for the recipe directions.", "The parsley is in the oven."}

Action 2: **open oven**

Feedback: Opened.

Action 3: **take parsley**

Feedback: Taken.

Action 4: **ask Charlie how to cut the parsley**

Feedback: The parsley needs to be sliced. Use a knife to cut it.

Notebook: {"You find yourself . . . for the recipe directions.", "The parsley is in the oven.", "The parsley needs to be sliced. Use a knife to cut it."}

Action 5: **slice parsley**

Feedback: Sliced.

Done after 5 steps. Score 1/1.

Figure 5. An excerpt from a *Q-TextWorld* game.

visible to the agent right from the start (*e.g.*, on the table), or hidden inside some container that needs to be opened first (*e.g.*, in the fridge). The player can ask the oracle where it can find a particular object (*e.g.*, `Ask Charlie`[3] `where's hot pepper?`). In return, the oracle will indicate where the object can be found (*e.g.*, `Hot pepper is in the fridge.`).

Take [1/2] + Cut: This task extends **Take [1/2]** as the ingredients also need to be cut in the right way (*i.e.*, `chopped`, `sliced`, or `diced`). Each cutting type is achieved by a different action command (*i.e.*, `chop X`, `slice X`, or `dice X`) while the player is holding a knife in their inventory. The player can also ask the oracle how to process a particular ingredient (*e.g.*, `Ask Charlie how to cut the hot pepper?`[4]). In return, the oracle will indicate which type of cutting is needed (*e.g.*, `Hot pepper needs to be sliced, use a knife to cut it`). Note, in the reported games, the player always start with a kitchen knife in their inventory.

B. Modeling Details

In this section, we provide detailed information regarding our agents. In Appendix B.1, we describe our agent used for the *Q-BabyAI* environments. In Appendix B.2, we describe our agent used for the *Q-TextWorld* environments.

[3] In *Q-TextWorld*, the oracle is named Charlie.

[4] Nonessential words can be omitted, *e.g.*, `Ask Charlie how hot pepper?`

General information	
Word vocabulary size	835
# Function word ($\lvert V_{\text{func}} \rvert$)	7
# Adjective ($\lvert V_{\text{adj}} \rvert$)	38
# Noun ($\lvert V_{\text{noun}} \rvert$)	45
# holders	6
# ingredients	42
# cuttable ingredients	26
Take 1	
# recipes	42
# configurations	1242
Avg. instruction length	33.75 ± 0.44
Avg. walkthrough length	1.49 ± 0.50
Avg. nb. entities	8.63 ± 0.96
Avg. observation length	150.94 ± 65.43
Avg. valid actions per step	4.90 ± 1.29
Take 2	
# recipes	1722
# configurations	1332156
Avg. instruction length	36.48 ± 0.60
Avg. walkthrough length	3.01 ± 0.70
Avg. nb. entities	10.69 ± 1.22
Avg. observation length	141.76 ± 57.56
Avg. valid actions per step	5.83 ± 1.65
Take 1 Cut	
# recipes	17576
# configurations	1026
Avg. instruction length	33.80 ± 0.40
Avg. walkthrough length	2.50 ± 0.50
Avg. nb. entities	9.37 ± 0.84
Avg. observation length	143.00 ± 59.30
Avg. valid actions per step	5.45 ± 1.70
Take 2 Cut	
# recipes	274625000
# configurations	859620
Avg. instruction length	36.61 ± 0.55
Avg. walkthrough length	5.00 ± 0.71
Avg. nb. entities	11.67 ± 1.13
Avg. observation length	138.59 ± 50.04
Avg. valid actions per step	7.73 ± 2.54

Table 9. Statistics of the *Q-TextWorld* environment.

B.1. AFK— *Q-BabyAI*

Observation Encoder (f_{obs}): Following BabyAI (Chevalier-Boisvert et al., 2019), the environment observation o^{env} of *Q-BabyAI* is a $7 \times 7 \times 4$ symbolic observation that contains a partial and local egocentric view of the environment and the direction of the agent. To encode o^{env}, we use a convolutional neural network (CNN). Following Chevalier-Boisvert et al. (2019), the observation encoder consists of three convolutional layers. The first convolutional layer has 128 filters of size 8×8 and stride 8. The second and third convolutional layers have 128 filters of size 3×3 and stride 1. Batch normalization and ReLU unit are applied to the output of each layer. At the end, a 2D pooling layer with filter size 2×2 is applied to obtain the representation h_o of 256 dimensions.

Word Encoder (f_{note}): Following Chevalier-Boisvert et al. (2019), we use a gated recurrent unit (GRU) (Chung et al., 2014) to perform word encoding. Specifically, for each $v_i \in A_0$, we have $h_i = f_{\text{gru}}(v_i) \in \mathcal{R}^{|v_i| \times l}$, where $|v_i|$ is the number of words in v_i and $l = 128$ is the encoding dimension.

Aggregator (f_{att}): Following the No Query baseline (Chevalier-Boisvert et al., 2019), the aggregator consists of FiLM (Perez et al., 2018) modules, f_{FiLM}, followed by a long short term memory (LSTM) f_{LSTM} (Hochreiter & Schmidhuber, 1997). That is, $f_{\text{att}} = f_{\text{LSTM}} \circ f_{\text{FiLM}}$. Specifically, we stack two FiLM modules. Each FiLM module has 128 filters with size 3×3 and the output dimension is 128. The LSTM has 128 units.

Physical Action and Query Heads ($\pi_{\text{switch}}, \pi_{\text{phy}}, \pi_{\text{fun}}, \pi_{\text{adj}}, \pi_{\text{noun}}$): The switch head π_{switch}, physical action head π_{phy}, and function word head π_{fun} are two-layer MLPs with 64 units in each layer. The output dimension of $\pi_{\text{switch}}, \pi_{\text{phy}}, \pi_{\text{fun}}$ are $2, 7, 2$. π_{adj} and π_{noun} are single-head pointer networks (Sec. 3.2) with hidden dimension $l = 128$.

B.2. AFK— *Q-TextWorld*

Text Encoder (f_{obs}, f_{note}): Due to the nature of the *Q-TextWorld* environment, where all inputs are in pure text, we share the two encoders (*i.e.*, f_{obs} and f_{note}) in our text agent.

We use a transformer-based text encoder, which consists of an embedding layer and a transformer block (Vaswani et al., 2017). Specifically, we tokenize an input sentence (either a text observation or an entry in the notebook) with the spaCy tokenizer.[5] We convert the sequence of tokens into 128-dimensional embeddings, the embedding matrix is initialized randomly.

The transformer block consists of a stack of 4 convolutional layers, a self-attention layer, and a 2-layer MLP with a ReLU non-linear activation function in between. Within the block, each convolutional layer has 128 filters, with a kernel size of 7. The self-attention layers use a block hidden size of 128, with 4 attention heads. Layer normalization (Ba et al., 2016) is applied after each layer inside the block. We merge positional embeddings into each block's input.

Given an input $o \in \mathbb{R}^{|o|}$, where $|o|$ denotes the number of tokens in o, the encoder produces a representation $h_o \in \mathbb{R}^{|o| \times H}$, with $H = 128$ the hidden size.

[5] https://spacy.io/

In practice, we use mini-batches to parallelize the training. Following standard NLP methods, we use special padding tokens when the number of tokens within a batch are different, we use masks to prevent the model from taking the padding tokens into computation. A text input o will be associated with a mask $m_o \in \mathbb{R}^{|o|}$.

Note for all the three agent variants (*i.e.*, No Query, Query Baseline and AFK), we use the concatenation of [*feedback*, *description*, *inventory*] as the input to f_{obs}. See examples of *feedback*, *description* and *inventory* text in Fig. 5.

Aggregator (f_{att}): To aggregate two input encodings $P \in \mathbb{R}^{|P|\times H}$ and $Q \in \mathbb{R}^{|Q|\times H}$, we use the standard multi-head attention mechanism (Vaswani et al., 2017). Specifically, we use P as the *query*, Q as the *key* and *value*. This results in an output $P_Q \in \mathbb{R}^{|P|\times H}$, where at every time step $i \in [0, |P|)$, P_Q^i is the weighted sum of Q, the weight is the attention of P^i on Q. We refer readers to Vaswani et al. (2017) for detailed information.

We apply a residual connection on top of the multi-head attention mechanism in order to maintain the original information contained in P. Specifically,

$$h_{PQ} = \text{Tanh}(\text{Linear}([P_Q; P])), \tag{4}$$

where $h_{PQ} \in \mathbb{R}^{|P|\times H}$, brackets $[\cdot;\cdot]$ denote vector concatenation.

We denote the above attention layer as

$$h_{PQ} = \text{Attention}(P, Q). \tag{5}$$

Using two of such layers (without sharing parameters), we aggregate three inputs: $h_{\text{obs}} \in \mathbb{R}^{|\text{obs}|\times H}$, $h_{\text{task}} \in \mathbb{R}^{|\text{task}|\times H}$ and $h_s \in \mathbb{R}^{|\text{note}|\times H}$, where $|\text{obs}|$, $|\text{task}|$ and $|\text{note}|$ denote the number of tokens in a text observation, the number of tokens in the instruction, and the number of nodes in the notebook:

$$\begin{aligned} h_{\text{obs,task}} &= \text{Attention}(h_{\text{obs}}, h_{\text{task}}), \\ h_x &= \text{Attention}(h_{\text{obs,task}}, h_s). \end{aligned} \tag{6}$$

Here, $h_{\text{obs,task}} \in \mathbb{R}^{|\text{obs}|\times H}$, $h_x \in \mathbb{R}^{|\text{obs}|\times H}$.

Action Generator (π_{func}, π_{adj}, π_{noun}): In *Q-TextWorld*, all actions follow the same format of `<func, adj, noun>`. Therefore, the query action space $\mathcal{A}_q$ and the physical action space $\mathcal{A}_{\text{phy}}$ are shared (*i.e.*, the vocabularies are shared). We use a three-head module to generate three vectors. Their sizes correspond to the function word, adjective, and noun vocabularies. The generated vectors are used to compute an overall Q-value.

Taking the aggregated representation $h_x \in \mathbb{R}^{|\text{obs}|\times H}$ as input, we first compute its masked average, using the mask of the text observation. This results in $\overline{h_s} \in \mathbb{R}^H$.

Specifically, the action generator consists of four multi-layer perceptrons (MLPs):

$$\begin{aligned} h_{\text{shared}} &= \text{ReLU}(\text{Linear}_{\text{shared}}(\overline{h_s})), \\ Q_{\text{func}} &= \text{Linear}_{\text{func}}(h_{\text{shared}}), \\ Q_{\text{adj}} &= \text{Linear}_{\text{adj}}(h_{\text{shared}}), \\ Q_{\text{noun}} &= \text{Linear}_{\text{noun}}(h_{\text{shared}}). \end{aligned} \tag{7}$$

Here, $Q_{\text{func}} \in \mathbb{R}^{|\text{func}|}$, $Q_{\text{adj}} \in \mathbb{R}^{|\text{adj}|}$, $Q_{\text{noun}} \in \mathbb{R}^{|\text{noun}|}$. $|\text{func}|$, $|\text{adj}|$, and $|\text{noun}|$ denote the vocabulary size of function words, adjectives, and nouns.

In order to alleviate the difficulties caused by a large action space, similar to the pointer mechanism in the *Q-BabyAI* agent, we apply masks over vocabularies when sampling actions. In the masks, only tokens appearing in the current notebook are labeled as 1, *i.e.*, the text agent only performs physical interaction with objects noted in its notebook. It also only asks questions about objects it has heard of.

Finally, we compute the Q-value of an action `<u, v, w>`:

$$Q_{<u,v,w>} = (Q_u + Q_v + Q_w)/3, \tag{8}$$

where u, v and w are tokens in the function word, adjective, and noun vocabulary.

C. Implementation Details

In this section, we provide implementation and training details of our agents. In Appendix C.1, we provide implementation details for our agent used for the *Q-BabyAI* environments. In Appendix C.2, we provide implementation details for our agent used for the *Q-TextWorld* environments.

C.1. AFK— *Q-BabyAI*

We closely follow the training procedure of the publicly available code of the BabyAI No Query agent (Chevalier-Boisvert et al., 2019). We train our AFK and all baselines with PPO (Schulman et al., 2017). Specifically, we use the Adam (Kingma & Ba, 2015) optimizer with learning rate 0.0001. We update the model every 2560 environment steps. The batch size is 1280. The PPO epoch is 4 and the discount factor is 0.99. We use 64 parallel processes for collecting data from the environment. The scaling factor β of the episodic exploration bonus is set to 0.1 for all experiments. For all experiments, we study uni-gram and bi-gram similarity models and report the better results. We tuned the episodic-exploration scaling factor $\beta \in \{0.001, 0.01, 0.1, 0.5\}$, hidden size of the pointer network $l \in \{32, 64, 128, 256\}$, learning rate $\in \{10^{-5}, 10^{-4}, 10^{-3}\}$, and similarity function $\in \{\text{uni-gram}, \text{bi-gram}\}$. We train all agents with 5 different random seeds: $[24, 42, 123, 321, 3407]$.

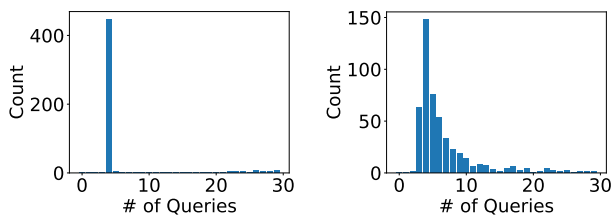


Figure 6. Evaluation episode count over number of queries. **Left**: train on ♣♠, evaluate on ♣♠. **Right**: train on ♠♦+♦♣, evaluate on ♣♠. ♣: **Obj. in Box**, ♠: **Danger**, ♦: **Go to Favorite**.

C.2. AFK— *Q-TextWorld*

We adopt the training procedure from the official code base released by TextWorld authors (Côté et al., 2018). Our text agent is trained with Deep Q-Learning (Mnih et al., 2013). We use a prioritized replay buffer with memory size of $500,000$, and a priority fraction of 0.5. During model update, we use a replay batch size of 64. We use a discount factor $\gamma = 0.9$. We use noisy nets, with a σ_0 of 0.5. We update the target network after every 1000 episodes. We sample the multi-step return $n \sim \text{Uniform}[1, 3]$. We refer readers to Hessel et al. (2018) for more information about different components of DQN training.

For all experiments, we use *Adam* (Kingma & Ba, 2015) as the optimizer. The learning rate is set to 0.00025 with a clip gradient norm of 5. We train all agents with 5 different random seeds: $[24, 42, 123, 321, 3407]$. For replay buffer data collection, we use a batch size of 20. We train our agents with 500K episodes, each episode has a maximum number of steps 20. After every 2 data collection steps, we randomly sample a batch from the replay buffer, and perform a network update.

Resources: We use a mixture of Nvidia V100, P100 and P40 GPUs to conduct all the experiments. On average, experiments on *Q-BabyAI* take 1 day, experiments on *Q-TextWorld* take 2 days.

D. Additional Results

Success rate and episode length: In Tab. 10 we report success rate and episode length of No Query, Query Baseline, and AFK on all levels of *Q-BabyAI* tasks. Note, due to the early termination mechanism, the comparison of episode length is only meaningful when the agent is able to solve the task. For instance, in **Object in Box** (♣) and **Danger** (♠), No Query and Query Baseline have shorter episode length than AFK because they either step on the danger tile or open the wrong box, resulting in the termination of an episode.

Number of queries made by an AFK agent in seen and unseen tasks: To better understand the agent's behavior in seen and unseen tasks, we report the number of queries an agent made across 500 evaluation episodes. As shown in Fig. 6 (left), when an agent is trained and evaluated on the same tasks (♣♠), in most episodes, the agent makes four queries, which is the optimal number of queries of the task. In contrast, when the agent is trained and evaluate on different tasks, Fig. 6 (right), it made more queries.

E. Training Curves

The training curves of all *Q-BabyAI* and *Q-TextWorld* experiments in terms of success rate and episode length are shown in Fig. 7, Fig. 8, Fig. 9, Fig. 10, and Fig. 11.

		No Query		Query Baseline		AFK (Ours)	
	Tasks	Succ.(%)	Eps. Len.	Succ.(%)	Eps. Len.	Succ.(%)	Eps. Len.
Lv. 1	♣	50.5±0.6	5.9±0.1	49.8±1.1	6.0±0.1	**100.0±0.0**	**10.8±0.1**
	♠	68.3±0.8	8.2±0.0	73.8±0.9	8.3±0.1	**100.0±0.0**	**10.4±0.0**
	♦	98.9±0.4	30.2±1.5	99.3±0.2	26.7±1.5	**100.0±0.0**	**16.8±6.7**
	♥	99.7±0.2	26.2±0.9	85.3±1.1	36.8±1.0	**100.0±0.0**	**20.6±0.2**
Lv. 2	♣♠	0.0±0.0	43.3±0.7	0.0±0.0	44.1±0.6	**90.3±1.8**	**15.5±0.3**
	♣♦	0.1±0.1	224.8±0.4	0.6±0.5	224.3±0.5	**94.3±2.3**	**18.8±0.3**
	♣♥	0.0±0.0	98.0±0.0	0.0±0.0	98.0±0.0	**99.0±0.4**	**32.3±0.6**
	♠♦	0.4±0.1	90.8±1.3	0.2±0.2	91.4±1.1	**100.0±0.0**	**14.5±0.0**
	♠♥	0.0±0.0	76.8±2.1	0.0±0.0	79.5±1.7	**0.0±0.0**	**79.0±0.9**
	♦♥	10.8±1.6	202.9±4.0	10.2±2.1	203.2±5.1	**98.7±0.2**	**66.9±3.2**
Lv. 3	♣♠♦	0.0±0.0	91.8±1.1	0.0±0.0	92.2±0.6	0.15±0.2	85.8±1.4
	♣♠♥	0.0±0.0	93.9±3.4	0.0±0.0	102.5±3.5	0.0±0.0	109.4±2.3
	♣♦♥	0.0±0.0	225.0±0.0	0.0±0.0	225.0±0.0	2.1±0.8	220.9±1.1
	♠♦♥	4.3±1.0	99.3±2.9	4.4±0.8	97.7±3.0	4.8±0.9	105.0±2.0
Lv. 4	♣♠♦♥	0.0±0.0	96.6±8.1	0.0±0.0	150.5±7.5	0.0±0.1	177.2±9.2

Table 10. Evaluation success rate and episode length on *Q-BabyAI*. ♣: **Object in Box**, ♠: **Danger**, ♦: **Go to Favorite**, ♥: **Open Door**.

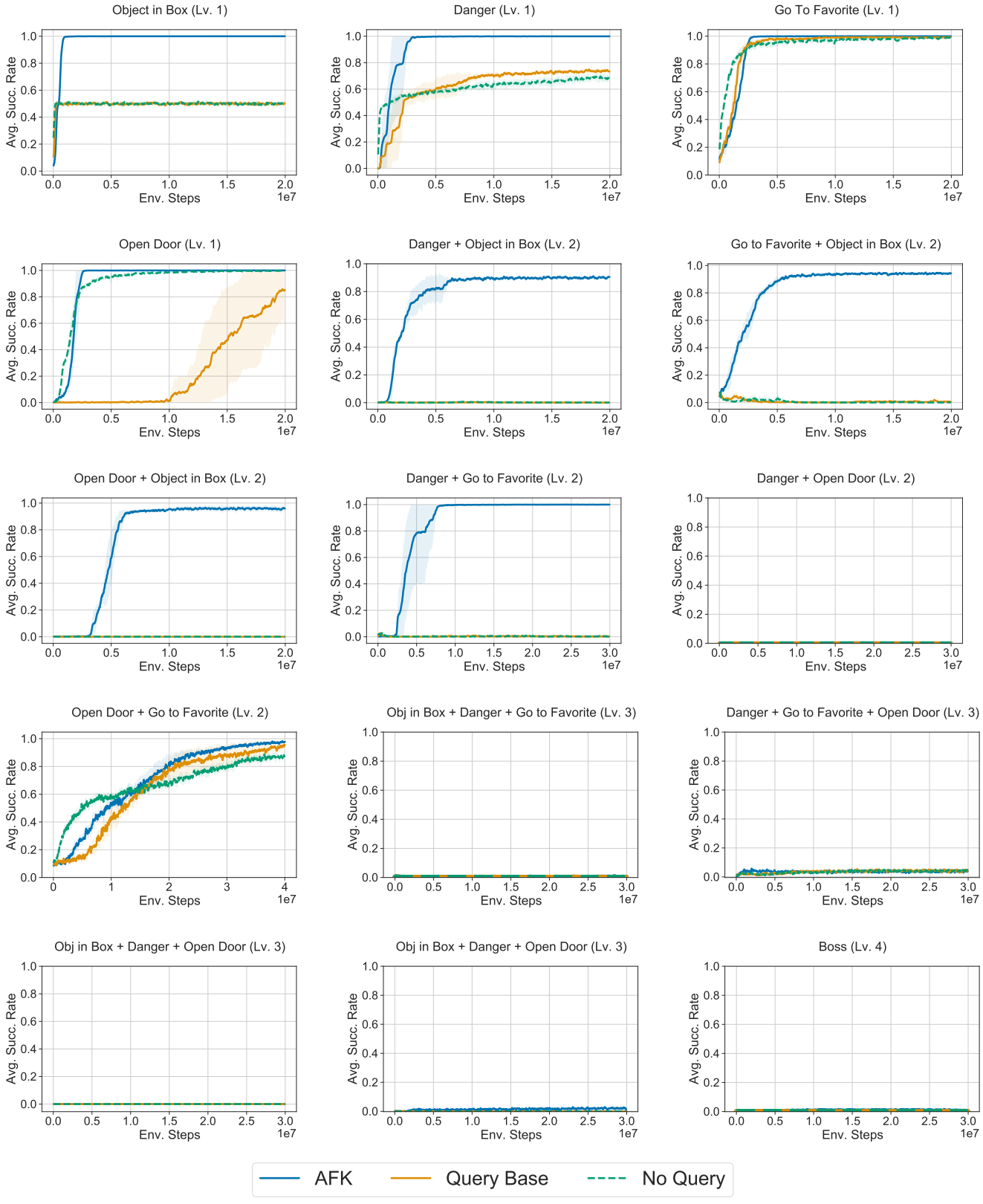


Figure 7. Success rate of AFK and baselines on *Q-BabyAI*.

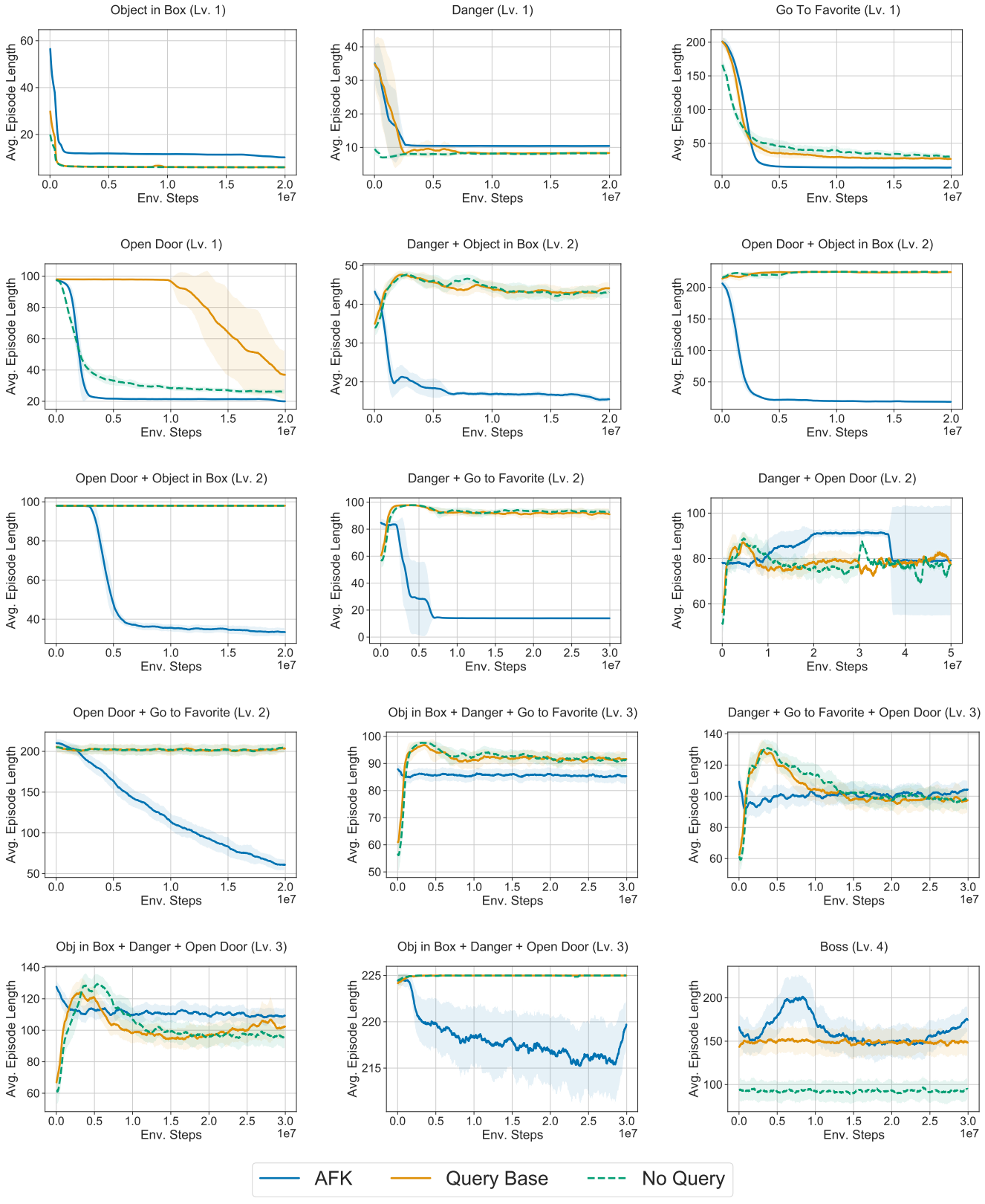


Figure 8. Episode length of AFK and baselines on *Q-BabyAI*.

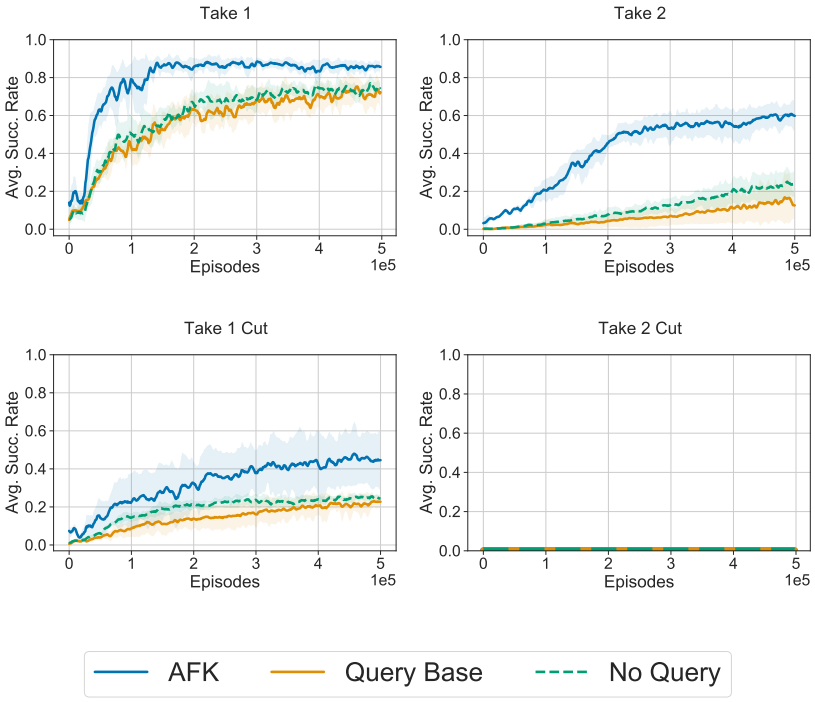


Figure 9. Success rate of AFK and baselines on *Q-TextWorld*.

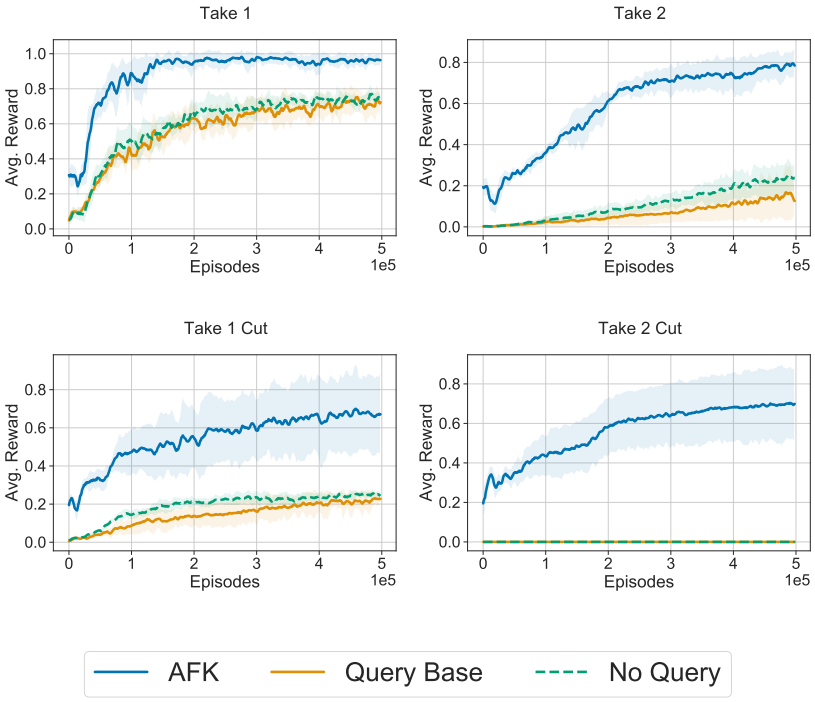


Figure 10. Training reward received by AFK and baselines on *Q-TextWorld*. Note, training reward is the sum of 1) reward given by the environment for solving the task; and 2) the episodic exploration bonus.

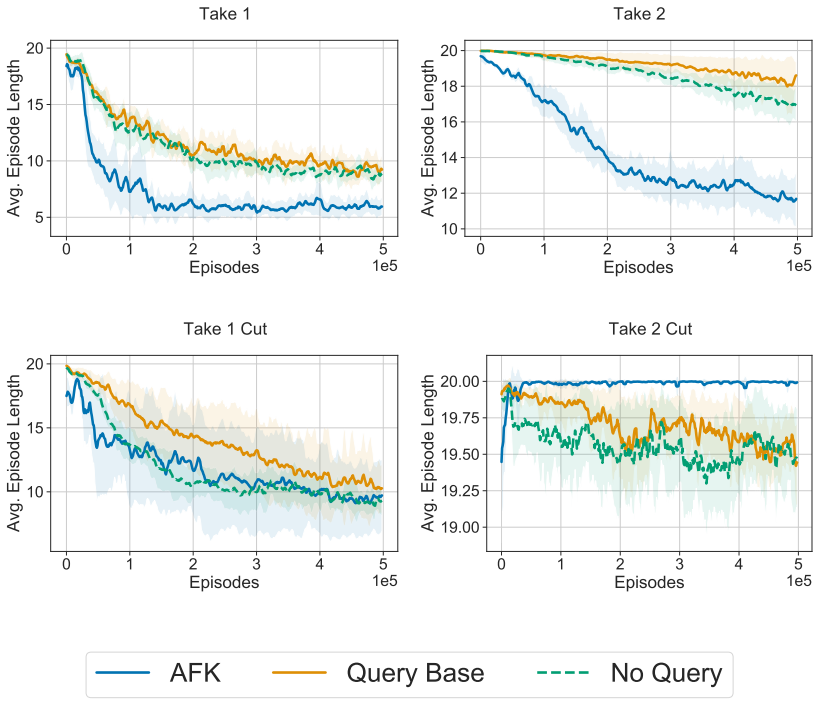


Figure 11. Steps used by AFK and baselines during training on *Q-TextWorld*.