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† 2 3 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.

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...
Asking for Knowledge (AFK)

query, we introduce two environments: the grid-world-based \textit{Q-BabyAI}, illustrated in Fig. 1 and inspired by BabyAI (Chevalier-Boisvert et al., 2019), and the text-based \textit{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 \textit{Q-BabyAI} and \textit{Q-TextWorld}, the knowledge source is designed to be task-agnostic, \textit{i.e.}, it replies to all queries, even if irrelevant to the task at hand. This mimics many real-world knowledge sources, \textit{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 cogni

![Image of a kitchen]  

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

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

\subsection{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 &
Asking for Knowledge (AFK)

Formally, a POMDP is defined by a tuple \((S, A, Z, T, O, R, \gamma, H)\). \(S\) is the state space, \(A\) is the action space, \(Z\) is the observation space. At each time step, the agent receives an observation \(o_t \in Z\) following the observation function \(O : S \rightarrow Z\) and selects an action \(a_t \in A\). The transition function \(T\) maps the action \(a_t\) and the current state \(s_t\) to a distribution over the next state \(s_{t+1}\), i.e., \(T : S \times A \rightarrow \Delta(S)\). The agent receives a real-valued reward \(r_t\) according to a reward function \(R : S \times A \rightarrow \mathbb{R}\). The agent’s goal is to maximize the return \(\sum_{t=0}^{H} \gamma^t r_t\), where \(\gamma\) 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 \(Z = Z_{env} \times Z_q\) is composed of \(Z_{env}\) and \(Z_q\), representing the agent’s surrounding and the response to a query, respectively.

Similarly, at each step, the agent’s action space \(A = A_{phy} \cup A_q\) is composed of the physical action space \(A_{phy}\) supported by classical RL environments (e.g., navigational actions, toggle, grasp) and the query action space \(A_q\).

Response Space \(Z_q\) and Query Action Space \(A_q\): 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 compositional 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 \(<\text{func}, \text{adj}, \text{noun}>\). In this 3-tuple, \text{func} is a function word selected from words like \textit{where’s}, \textit{what’s} and \textit{how’s}, which indicates the function of a query (e.g., inquire about an object’s location or affordances). The combination of an adjective (\text{adj}) and a \text{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 \(<\text{func}, \text{adj}, \text{noun}>\) 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-
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 \( o_t^{\text{env}} \). Moreover, if the agent issued a query at time step \( t - 1 \), it also receives a language response \( v_t \in Z_q \) from the oracle, otherwise \( v_t = \emptyset \).

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 \( o_t^{\text{env}} \) 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 \( \pi_{\text{switch}}(\cdot|h_x) : \mathbb{R}^l \rightarrow \Delta(\{0, 1\}) \), a physical action head \( \pi_{\text{phys}}(\cdot|h_x) : \mathbb{R}^l \rightarrow \Delta(\mathcal{A}_{\text{phys}}) \), a function word head: \( \pi_{\text{func}}(\cdot|h_x) : \mathbb{R}^l \rightarrow \Delta(\mathcal{V}_{\text{func}}) \), an adjec-
At the beginning of each episode, the notebook gathers information related to the task at hand. The AFK agent only considers the set \( A_0 \) which contains the task instruction \( v_0 \). To address the challenge of a combinatorially growing action space, we develop a pointer mechanism for the policies \( \pi_{\text{adj}} \) and \( \pi_{\text{noun}} \). 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 \( \pi_{\text{adj}} \) and \( \pi_{\text{noun}} \), i.e., \( \pi_{\text{adj}} \) and \( \pi_{\text{noun}} \) 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 \( m_{\text{noun}} \) denote the number of nouns in \( A_0 \), and let \( h_w \in \mathbb{R}^{m_{\text{noun}} \times d} \) 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

\[
q = h_x \cdot W_q, \quad \text{and} \quad k = h_w \cdot W_k,
\]

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

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

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

<table>
<thead>
<tr>
<th>Tasks</th>
<th>No Query</th>
<th>Query Baseline</th>
<th>AFK (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lv. 1</td>
<td>✷</td>
<td>50.5±2.0</td>
<td>49.8±1.2</td>
</tr>
<tr>
<td></td>
<td>♥</td>
<td>68.3±2.4</td>
<td>73.8±1.2</td>
</tr>
<tr>
<td></td>
<td>♦</td>
<td>98.9±0.8</td>
<td>99.3±0.3</td>
</tr>
<tr>
<td></td>
<td>♣♠</td>
<td>99.7±0.3</td>
<td>85.3±22.3</td>
</tr>
</tbody>
</table>

| 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.

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 $r^\text{env}$ is the external reward and $b$ is the bonus reward. A positive bonus reward $b$ is obtained whenever a query’s response $v_i \not\in \varnothing$ expands the agent’s knowledge about the task, i.e., $A_0$. The reward is only given for new information. Formally,

$$b = \beta I[(v_i \in A_0) \land (v_i \not\in A_0^t)],$$

where $v_i$ denotes a new response returned by the oracle, and $A_0^t$ 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 pro-

$^1$BabyAI: github:mila-iqia/babyai
$^2$TextWorld: github:xingdi-eric-yuan/qait_public

posed agent via AFK, which is the agent with 1) notebook, 2) pointer mechanism, and 3) episodic exploration.

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).

<table>
<thead>
<tr>
<th>Target Task</th>
<th>Source Tasks</th>
<th>Succ. (%)</th>
<th>Eps. Len.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Danger</td>
<td>♠♥</td>
<td>22.1±0.7</td>
<td>32.0±0.3</td>
</tr>
<tr>
<td>Obj in Box + Open Door</td>
<td>♠♦</td>
<td>35.6±2.1</td>
<td>71.6±1.1</td>
</tr>
</tbody>
</table>

Table 5. Zero-shot generalization study of AFK.

| Task         | \(|Q|\) | Precision | Recall | F1    |
|--------------|-------|-----------|--------|-------|
| Go to Favorite | ♦     | 0.804     | 0.823  | 0.812 |
| Danger       | ♣     | 0.771     | 0.560  | 0.601 |
| Go to Favorite | ♦     | 0.989     | 0.989  | 0.989 |
| Open Door    | ♣     | 0.989     | 0.989  | 0.989 |

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

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 \(A_q\)), 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 task is associated with a set of queries $Q_i$ which are useful for solving the task. If an agent issues a query $q \in Q_i$, 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 $= \frac{n_g}{n_{tot}}$, recall $= \frac{n_g}{|Q_i|}$, 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


Asking for Knowledge (AFK)


Appendix: Asking for Knowledge (AFK)

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

<table>
<thead>
<tr>
<th>General information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Word vocabulary size</td>
<td>62</td>
</tr>
<tr>
<td>Function word vocabulary size ($</td>
<td>V_{func}</td>
</tr>
<tr>
<td>Adjective vocabulary size ($</td>
<td>V_{adj}</td>
</tr>
<tr>
<td>Noun vocabulary size ($</td>
<td>V_{noun}</td>
</tr>
<tr>
<td># of Physical action</td>
<td>7</td>
</tr>
<tr>
<td>Visual range</td>
<td>$7 \times 7$</td>
</tr>
<tr>
<td># of object colors</td>
<td>6</td>
</tr>
<tr>
<td># of actionable object types</td>
<td>4</td>
</tr>
<tr>
<td># of names</td>
<td>2</td>
</tr>
<tr>
<td># of danger zone colors</td>
<td>2</td>
</tr>
</tbody>
</table>

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 $|Q_i|$ 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. 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
Asking for Knowledge

Tasks

<table>
<thead>
<tr>
<th>Qt</th>
<th># of rooms</th>
<th>room size</th>
<th>Early Terminate</th>
</tr>
</thead>
<tbody>
<tr>
<td>♣</td>
<td>3</td>
<td>9×9</td>
<td>True</td>
</tr>
<tr>
<td>♠</td>
<td>1</td>
<td>7×7</td>
<td>True</td>
</tr>
<tr>
<td>♦</td>
<td>2</td>
<td>5×5</td>
<td>False</td>
</tr>
<tr>
<td>♥</td>
<td>1</td>
<td>7×7</td>
<td>False</td>
</tr>
<tr>
<td>Lv. 1</td>
<td>♣♠</td>
<td>5</td>
<td>7×7</td>
</tr>
<tr>
<td>♣♦</td>
<td>4</td>
<td>5×5</td>
<td>True</td>
</tr>
<tr>
<td>♣♥</td>
<td>5</td>
<td>7×7</td>
<td>True</td>
</tr>
<tr>
<td>♠♦</td>
<td>3</td>
<td>7×7</td>
<td>True</td>
</tr>
<tr>
<td>♠♥</td>
<td>2</td>
<td>7×7</td>
<td>True</td>
</tr>
<tr>
<td>♦♥</td>
<td>4</td>
<td>5×5</td>
<td>False</td>
</tr>
<tr>
<td>Lv. 2</td>
<td>♣♠♥</td>
<td>6</td>
<td>7×7</td>
</tr>
<tr>
<td>♣♦♥</td>
<td>5</td>
<td>5×5</td>
<td>True</td>
</tr>
<tr>
<td>♣♦♥</td>
<td>4</td>
<td>7×7</td>
<td>True</td>
</tr>
<tr>
<td>Lv. 3</td>
<td>♣♠♦♥</td>
<td>7</td>
<td>7×7</td>
</tr>
<tr>
<td>Lv. 4</td>
<td>♣♠♦♥</td>
<td>7</td>
<td>7×7</td>
</tr>
</tbody>
</table>

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.

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.

Action 5: slice parsley

Feedback: Sliced.

Done after 5 steps. Score 1/1.

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

---

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.

---

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

4Nonessential words can be omitted, e.g., Ask Charlie how hot pepper?
Table 9. Statistics of the Q-TextWorld environment.

<table>
<thead>
<tr>
<th>General information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Word vocabulary size</td>
<td>835</td>
</tr>
<tr>
<td># Function word ($</td>
<td>7</td>
</tr>
<tr>
<td># Adjective ($V_{adj}$)</td>
<td>38</td>
</tr>
<tr>
<td># Noun ($V_{noun}$)</td>
<td>45</td>
</tr>
<tr>
<td># holders</td>
<td>6</td>
</tr>
<tr>
<td># ingredients</td>
<td>42</td>
</tr>
<tr>
<td># cuttable ingredients</td>
<td>26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Take 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># recipes</td>
<td>42</td>
</tr>
<tr>
<td># configurations</td>
<td>1242</td>
</tr>
<tr>
<td>Avg. instruction length</td>
<td>33.75 ± 0.44</td>
</tr>
<tr>
<td>Avg. walkthrough length</td>
<td>1.49 ± 0.50</td>
</tr>
<tr>
<td>Avg. nb. entities</td>
<td>8.63 ± 0.96</td>
</tr>
<tr>
<td>Avg. observation length</td>
<td>150.94 ± 65.43</td>
</tr>
<tr>
<td>Avg. valid actions per step</td>
<td>4.90 ± 1.29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Take 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># recipes</td>
<td>1722</td>
</tr>
<tr>
<td># configurations</td>
<td>1332156</td>
</tr>
<tr>
<td>Avg. instruction length</td>
<td>36.48 ± 0.60</td>
</tr>
<tr>
<td>Avg. walkthrough length</td>
<td>3.01 ± 0.70</td>
</tr>
<tr>
<td>Avg. nb. entities</td>
<td>10.69 ± 1.22</td>
</tr>
<tr>
<td>Avg. observation length</td>
<td>141.76 ± 57.56</td>
</tr>
<tr>
<td>Avg. valid actions per step</td>
<td>5.83 ± 1.65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Take 1 Cut</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># recipes</td>
<td>17576</td>
</tr>
<tr>
<td># configurations</td>
<td>1026</td>
</tr>
<tr>
<td>Avg. instruction length</td>
<td>33.80 ± 0.40</td>
</tr>
<tr>
<td>Avg. walkthrough length</td>
<td>2.50 ± 0.50</td>
</tr>
<tr>
<td>Avg. nb. entities</td>
<td>9.37 ± 0.84</td>
</tr>
<tr>
<td>Avg. observation length</td>
<td>143.00 ± 59.30</td>
</tr>
<tr>
<td>Avg. valid actions per step</td>
<td>5.45 ± 1.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Take 2 Cut</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># recipes</td>
<td>274625000</td>
</tr>
<tr>
<td># configurations</td>
<td>859620</td>
</tr>
<tr>
<td>Avg. instruction length</td>
<td>36.61 ± 0.55</td>
</tr>
<tr>
<td>Avg. walkthrough length</td>
<td>5.00 ± 0.71</td>
</tr>
<tr>
<td>Avg. nb. entities</td>
<td>11.67 ± 1.13</td>
</tr>
<tr>
<td>Avg. observation length</td>
<td>138.59 ± 50.04</td>
</tr>
<tr>
<td>Avg. valid actions per step</td>
<td>7.73 ± 2.54</td>
</tr>
</tbody>
</table>

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 $8 \times 8$ and stride 8. The second and third convolutional layers have 128 filters of size $3 \times 3$ 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 $2 \times 2$ 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 $v_i \in A_o$, we have $h_i = f_{gru}(v_i) \in R^{|v_i| \times l}$, 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, $f_{att} = f_{LSTM} \circ f_{FiLM}$. Specifically, we stack two FiLM modules. Each FiLM module has 128 filters with size $3 \times 3$ and the output dimension is 128. The LSTM has 128 units.

Physical Action and Query Heads ($\pi_{switch}$, $\pi_{phy}$, $\pi_{func}$, $\pi_{adj}$, $\pi_{noun}$): The switch head $\pi_{switch}$, physical action head $\pi_{phy}$, and function word head $\pi_{func}$ are two-layer MLPs with 64 units in each layer. The output dimension of $\pi_{switch}$, $\pi_{phy}$, and $\pi_{func}$ are 2, 7, 2. $\pi_{adj}$ and $\pi_{noun}$ 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. 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 $o \in R^{|o|}$, where $|o|$ denotes the number of tokens in $o$, the encoder produces a representation $h_o \in R^{|o| \times H}$, with $H = 128$ the hidden size.

5https://spacy.io/
We apply a residual connection on top of the 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 $i \in [0, |P|)$, $P_Q$ 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,

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

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

We denote the above attention layer as

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

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:

$$h_{\text{obs, task}} = \text{Attention}(h_{\text{obs}}, h_{\text{task}}),$$

$$h_x = \text{Attention}(h_{\text{obs, task}}, h_s). \quad (6)$$

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

**Action Generator ($\pi_{\text{func}}, \pi_{\text{adj}}, \pi_{\text{noun}}$):** In $Q$-TextWorld, all actions follow the same format of $\langle \text{func}, \text{adj}, \text{noun} \rangle$. Therefore, the query action space $A_{\text{phy}}$ and the physical action space $A_{\text{phy}}$ 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 $h_x \in \mathbb{R}^H$.

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

$$h_{\text{shared}} = \text{ReLU}(\text{Linear}_{\text{shared}}(h_s)),$$

$$Q_{\text{func}} = \text{Linear}_{\text{func}}(h_{\text{shared}}),$$

$$Q_{\text{adj}} = \text{Linear}_{\text{adj}}(h_{\text{shared}}),$$

$$Q_{\text{noun}} = \text{Linear}_{\text{noun}}(h_{\text{shared}}).$$

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 $\langle u, v, w \rangle$:

$$Q_{\langle u, v, w \rangle} = (Q_u + Q_v + Q_w)/3, \quad (8)$$

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 $\beta$ 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 $\in \{$uni-gram, bi-gram$\}$. We train all agents with 5 different random seeds: [24, 42, 123, 321, 3407].
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 $\sigma_0 = 0.5$. 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.
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.
Asking for Knowledge (AFK)

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.
Asking for Knowledge (AFK)

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