EAGER: Asking and Answering Questions for Automatic Reward Shaping in Language-guided RL

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reinforcement learning (RL) in long horizon and sparse reward tasks is notoriously difficult and requires a lot of training steps. A standard solution to speed up the process is to leverage additional reward signals, shaping it to better guide the learning process. In the context of language-conditioned RL, the abstraction and generalisation properties of the language input provide opportunities for more efficient ways of shaping the reward. In this paper, we leverage this idea and propose an automated reward shaping method where the agent extracts auxiliary objectives from the general language goal. These auxiliary objectives use a question generation (QG) and a question answering (QA) system: they consist of questions leading the agent to try to reconstruct partial information about the global goal using its own trajectory. When it succeeds, it receives an intrinsic reward proportional to its confidence in its answer. This incentivizes the agent to generate trajectories which unambiguously explain various aspects of the general language goal. Our experimental study shows that this approach, which does not require engineer intervention to design the auxiliary objectives, improves sample efficiency by effectively directing the exploration.

1 Introduction

One of the main challenges of Reinforcement Learning (**RL**) research is to train agents able of abstraction, generalisation and communication. Languages, be they natural or formal, afford these desirable properties [11]. Based on this insight, many papers have tried to leverage the abilities of language in RL to enable communication and improve generalisation and sample efficiency [3, 21, 12, 35]. The domain can be subdivided into language-conditioned RL (LC-RL), in which language conditions the formulation of the problem[2, 12], and language-assisted RL, where language helps the agent to learn [15, 10, 1, 9]. In the present paper, we focus on the LC-RL framework where the agent initially receives a language instruction and must act to optimise the corresponding reward function. Unfortunately, the corresponding RL algorithms are sample inefficient, especially due to the fact that the reward function is sparse when it is restricted to completing the goal.

To tackle the reward sparsity issue, one idea is to densify the reward by decomposing the general goal into sub-goals and rewarding them individually. This idea is based on a decomposition principle which hypothesises that a general goal can be decomposed into a set of easier ones. Previous works assume a strict decomposition of the general goal into sub-goals [3, 17]. However, strict decomposition requires that that each high-level goal can be reached through an exact series of low-level policies, an assumption which fails when all the primitive actions are not known in advance. More recently, the ELLA method was proposed as a language reward shaping technique that is sample efficient and does not require a strict decomposition: it rewards interesting auxiliary objectives, without requiring rigid ordering and correspondence with sub-goals [22].

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

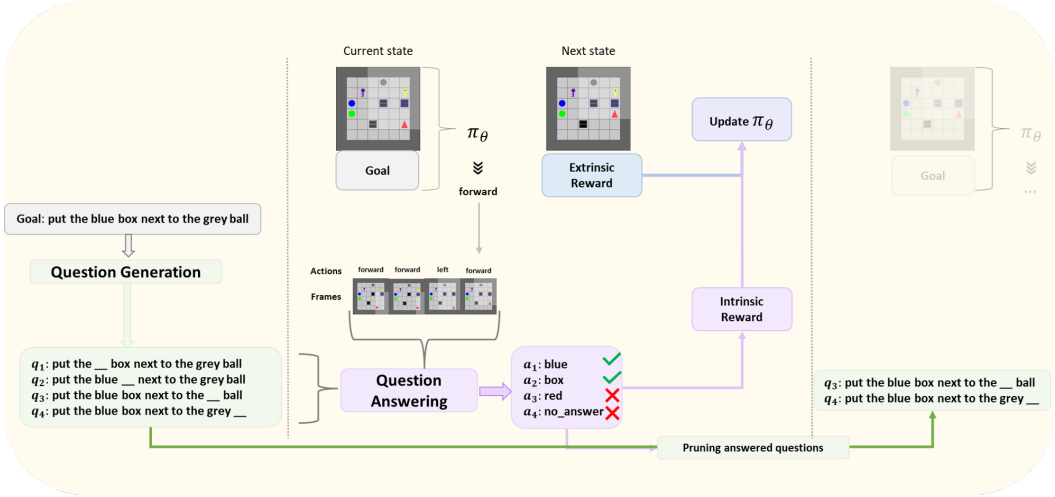

Figure 1: During training, the agent uses the goal to generate relevant questions using its question-generation module **QG**. Then, it attempts at answering them from current trajectories at each step with its question-answering module **QA**, by looking at the trajectory. When it succeeds, it obtains an intrinsic reward proportional to its confidence in its answer. Then it removes the answered questions from the list of questions. This incentivizes the agent to produce trajectories that enable to reconstruct unambiguously partial information about the general language goal, enabling to shape rewards and guide learning.

However, all these methods suffer from the need for expert input. Indeed, they require task-specific engineering for determining at least one of the following elements: the set of sub-goals or auxiliary objectives, the relevant ones and when they are achieved, and the appropriate reward. While all these points can be addressed more or less easily in simple game-like environments, they get demanding when the environment is more complex. Therefore, it seems desirable to find an alternative reward shaping approach that minimises expert involvement. We would like the agent to generate its own auxiliary objectives and create intrinsic rewards, just like a human would do.

The Natural Language Processing (**NLP**) field often suffers from the same need of expert input, e.g. for the evaluation of automatic summary tasks. In that context, some successful approaches have developed reference-less metrics [30], [28], based on question generation (**QG**) and question answering (**QA**). These techniques assess the quality of a generated text by measuring the quantity of information from the source conserved in the generated text (broadly, how well one can answer questions about the original text using the generated text). These metrics are said reference-less because they do not require a comparison with a man-made example to evaluate the quality of a text.

Contributions In our work, we build on these reference-less metrics to circumvent the need of expert input for generating auxiliary objectives. We adapt it and propose a novel QG/QA framework for RL called EAGER[1]. In EAGER, an agent reuses the initial language goal sentence to generate a set of questions (QG): each of these self-generated questions defines an auxiliary objective. Here, generating a question consists in masking a word of the initial language goal. Then the agent tries to answer these questions (guess the missing word) only by observing its trajectory so far. When it manages to answer a question correctly (QA) it obtains an intrinsic reward proportional to its confidence in the answer. The sum of all the intrinsic rewards measures the quality of a trajectory in relation to the given goal. In other words, maximizing this intrinsic reward incentivizes the agent to produce behaviour that unambiguously explains various aspects of the given goal.

To the best of our knowledge, EAGER is the only framework that can automatically 1) generate relevant auxiliary objectives, 2) determine their completion and 3) return the appropriate intrinsic reward. This approach only assumes the agent has access to a dataset of demonstrated behaviours associated to global language commands, which enables it to pre-train its question answering module.

[1]Exploit question-Answering Grounding for effective Exploration in language-conditioned Reinforcement learning, see `https://anonymous.4open.science/r/EAGER-FC2E/README.md` for access to the code.

Thus our work brings the following contributions:

- We create a QG/QA metric providing to an agent an information-rich measure of the quality of its trajectory given a goal.
- We propose the EAGER framework that lets the agent guide its own learning process by generating auxiliary objectives and producing intrinsic rewards without requiring any expert intervention.
- We show that EAGER retains the good properties of ELLA without requiring task-specific expert knowledge, by leveraging properties of language.
- We experiment EAGER with the BabyAI platform [7]: we compare our approach against ELLA (SOTA on BabyAI) and RIDE (a non-language based reward shaping approach using intrinsic motivation), showing its robustness and sample efficiency.

2 Related work

Language-conditioned RL. We place our work in the LC-RL setting, where an agent learns a policy to execute language commands [23, 7, 19, 20]. We reuse the BabyAI platform [7], widely used in this domain as it enables to decouple exploration challenges from perception challenges. It uses a synthetic language exhibiting interesting combinatorial properties with possible conjunction of properties, and procedural generation to avoid overfitting [8]. Here, we consider *instruction following* agents which receive external instructions and rewards [14, 5, 17].

Language as an abstraction in hierarchical RL. Several approaches leverage language for abstraction in hierarchical RL. One approach uses language for training a low-level instruction-following policy, then learns a high-level policy that generates the sequence of low-level goals [17]. Another one explicitly decomposes high-level tasks into low-level ones as in policy sketches [3]. ELLA [22] also uses language to decompose high-level tasks but relaxes the strict decomposition constraint by replacing sub-goals with auxiliary objectives. We adopt the same flexible framework in our approach but automatise the decomposition.

Language for exploration and reward shaping. Reward shaping is a form of guidance that supplies additional rewards to the agent to direct its learning process. Among approaches studying how language can shape rewards and exploration, LEARN [12] proposes to map intermediate natural language instruction to intermediate rewards. Similarly, [34] enables reward shaping using natural language through a narration-guided method. The high-level tasks are decomposed into low-level tasks and rewarded using narration. ELLA [22] is positioned in the same paradigm but with fewer assumptions about the environment or the structure of the task.

Other approaches assume that an oracle provides language descriptions of environment states which are used as state abstraction to generate novelty intrinsic rewards and guide exploration [24, 33]. This extends classical approaches to intrinsically motivated exploration [4], such as count-based methods [6] or RIDE [27]. Here, we also use language to generate intrinsic rewards, but we do not need language descriptions of states. Besides, other forms of intrinsic motivation systems, such as IMAGINE [10], first learn language-parameterized reward functions through interaction with a social peer, then autonomously use language to generate diverse and novel goals, using the learned reward function for self-supervision.

Asking questions in RL. Beyond reward shaping, some methods consider agents that use language to ask questions to external knowledge sources. In QWA [35], questions are used to identify sub-tasks and prune irrelevant actions. In AFK [18], questions are used to obtain world knowledge that helps completing tasks.

Natural Language Processing One of the sources of QG/QA methods is the thriving field of question generation from natural language processing and information retrieval [16]. Our approach is inspired from text generation methods where QG and QA are used to measure the quality of a generated text without using a human reference [30, 28].

3 Problem statement

We set ourselves in the standard framework of LC-RL with an augmented Partial Observation Markov Decision Process [32] $\mathcal{M}$ defined by the tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{Z}$, $\mathcal{T}$, $\mathcal{O}$, r, $\mathcal{G}$, γ), with $\mathcal{S}$ the state space, $\mathcal{Z}$ the observation state, $\mathcal{A}$ the actions space, $\mathcal{G}$ the goal space. At each time step t, the agent receives an observation $o_t \in \mathcal{Z}$ following the observation function $\mathcal{O} : \mathcal{S} \rightarrow \mathcal{Z}$ and selects an action $a_t \in \mathcal{A}$ to reach a goal $g \in \mathcal{G}$. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$ is the transition function. R is the goal-conditioned state action reward function, with $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow [0, 1]$ the extrinsic reward received for some goal $g \in \mathcal{G}$ and γ the discount factor. Using RL, we search for an optimal goal-conditioned policy π^*, such that $\pi^* : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ maximises the discounted expected return $R_t = \mathbb{E}[\sum_{k=0}^{T} \gamma^k \, r_{t+k+1}]$. Besides, we assume a sparse reward and a limited number of steps H to complete the task, resulting in hard exploration. Using intrinsic rewards, we modify the global reward from r to r' through a policy invariant transformation [25], which ensure that π^* is optimal for both $\mathcal{M}$ and $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r', \mathcal{G}, \gamma)$.

We also assume that there exists an auxiliary objectives set $\mathcal{G}^*$ such that for any $g \in \mathcal{G}$ there exists a sub-set $\{g_1^*, ..., g_l^*\}$ in $\mathcal{G}^*$, whose accomplishment can help the agent reach g. These auxiliary objectives do not have to strictly decompose the goal g. For instance, *Go to bread* and *Go to knife* can help achieve *Cut some bread*, but these goals do not contain all the necessary actions. Crucially, not all $g^* \in \mathcal{G}^*$ are interesting to do a certain goal g. In previous work, selecting relevant goals required some expert input. In ELLA [22], such auxiliary objectives have been used to guide the agent via reward shaping. However, as the environment becomes more complex, $\mathcal{G}^*$ grows quickly. Moreover, the vast majority of the auxiliary objectives will not be interesting for a particular goal g. Thus, a way must be found to recover the relevant auxiliary objectives, find a way to measure their completion, and associate the appropriate reward. In EAGER we propose an automated way to do it.

4 Method

In this section, we first introduce EAGER, our automated reward shaping method based on QG/QA, then we present a practical implementation in the context of the BabyAI benchmark. Figure 1 provides a graphical overview of EAGER.

4.1 EAGER

We need an automatic evaluation method that is fine-grained enough to rank various trajectories depending on a language instruction. But various successful policies can generate a set of valid trajectories for the same goal. It could be deceiving to rank them based only on their final results, as an overly complex trajectory seems intuitively worse than a simpler one even if the result is the same.

To address this issue, the EAGER framework consists of an agent learning module, a question generation module QG (automatic, but not learned) and a learned question answering module QA. It is inspired from works like QuestEval [30] and Data-QuestEval [28], developed for natural language generation. For instance, for abstractive summarization, by generating questions from the original text (QG) and trying to answer them using the summary QA, this metric measures the quantity of information shared between both texts.

In our work, we draw the analogy with the NLP task with the goal replacing the original text and the trajectory replacing the summary. We use the QG/QA system to verify that a trajectory contains the same level of information as the language instruction, meaning that the goal is contained in the trajectory. As the goal can be contained in a lot of different trajectories, we also favour simple trajectories. If one can easily answer the question, that means the trajectory is simple.

The QG module We assume that goals $g \in \mathcal{G}$ are highly expressive (e.g *Put the red ball next to the blue box*, *open the red door*, ...), containing by themselves enough world knowledge to generate questions by masking words. Thus, the QG module returns a list of k questions $QG_k(g)$. Each question is formed by masking one word in the linguistic instruction. Crucially, the choice of words to mask can be done automatically without any expert knowledge of the task, or the environment, for instance masking all nouns and adjectives. These questions can be seen as auxiliary objectives guiding the agent during training. Besides, being formulated in natural language, these auxiliary objectives are easily interpreted.

The QA module Let $\tilde{A}$ be the set of possible answers generated automatically from the list of tokens masked by the QG. The QA module returns the probability for all $\tilde{a} \in \tilde{A}$, that $\tilde{a}$ answers question q, given a trajectory τ, where $\tau = (o_i, a_i)_{i \in [|0,t|]}$ are state-action pairs. We note $\tilde{a}^* = QA(q, \tau)$ the answer greedily generated from the module and $QA(\tilde{a}^* \mid q, \tau)$ the associated probability. The auxiliary objectives from the QG are considered achieved when the QA answers them successfully. In our work, the QA module is pre-trained using example trajectories generated by a bot, see Section 5.1, without any type of annotations to guide it. Besides, at time step t, there is no guarantee that the trajectories contain enough information to correctly answer a question. Thus we include a *«no_answer»* token in $\tilde{A}$ to prevent the QA from answering correctly by chance. Moreover, since the QA module takes the whole trajectory, once it has answered a question, it can also answer it at the next step. To avoid giving a reward that does not have direct link to the current step, every time a question q is answered, it is removed from the set of questions: We note $\mathcal{Q}_t$ the active set of questions, the initial set is $\mathcal{Q}_0 = QG_k(g)$ and once a question is answered, we apply $\mathcal{Q}_t \leftarrow \mathcal{Q}_{t-1} \setminus \{q\}$.

Architecture of the QA The QA is used to compositionally chain low-level tasks. To do this, it relies on the episodic transformer [26]. This architecture uses multimodal transformers (over language, visual observation, and a list of actions) that have access to the full episode. For any time step t and any question, a $(x_{1:L}, v_{1:t}, a_{1:t})$ tuple is given as input to the QA module. The language input $x_{1:L}$ is the question, it is a sequence of L tokens with $x_i \in \mathbb{N}$. The visual input $v_{1:t}$ is the list of observations $v_t \in \mathbb{R}^{W \times H \times 3}$. Finally, the action input $a_{1:t}$ is a list of discrete actions. As output, the network returns the probability distribution over the set of possible answers $\tilde{\mathcal{A}}$.

Intrinsic reward The main difficulty in ranking trajectories is that many trajectories can share the same description and thus can answer questions correctly. However, a human assessing several trajectories, with the same extrinsic reward, relatively to the same goal would prioritise the simplest one. To account for this, we make the reward proportional to the confidence in the answer: $QA(\tilde{a}^* \mid q, \tau)$. An overly complex trajectory being harder to understand for the QA, the answer should be given with less confidence, so it should obtain less reward than a direct and clear trajectory, even with the same number of correct answers.

In this paper, we keep the same desired properties for the shaped reward function as the ones given in ELLA: the reward shaping should not change the optimal policy that prevails before the reward shaping (policy invariance), and the reward should encourage sample efficient exploration based on auxiliary-objectives. To do so, we ensure that only successful trajectories get the same return with or without the reward shaping by subtracting the shaped reward at the final time step N of a successful trajectory: $r'_N = r_N - \sum_{t \in T_{\mathbb{S}}} \gamma^{t-N} r'_t$ with $r'_t = \lambda \sum_{q \in \mathcal{Q}_t} \sum_{\tilde{a}^* \textit{is correct answer}} QA(\tilde{a}^* \mid q, \tau_t)$, where $T_{\mathbb{S}}$ is the set of time steps where a bonus is applied. r_t is the reward given at time step t, in the case of sparse reward studied here $r_t \neq 0$ only if t corresponds to the last step of a successful trajectory. In ELLA Mirchandani et al. [22] the authors prove that this transformation is policy invariant. Thus, as long as the policy produces unsuccessful trajectories, the agent is guided by the shaping reward. Then once it has learn to successfully complete an instruction, the shaping reward is subtracted at the last step and the agent improves using only the extrinsic reward. Further details are provided in the appendix.

Please note that the learning of the QA system is done with successful trajectories. This prevents reward hacking. Indeed, if the agent individually complete the auxiliary objectives without a meaningful trajectory, the QA does not consider the trajectory meaningful and answers: "no_answer" preventing the agent to get rewarded.

Using the above notations and concepts, we can define a metric to measure the adequacy of a trajectory to a goal, that corresponds to our cumulative intrinsic reward over a trajectory τ of length N, up to a λ factor:

$$m_{QG/QA}(g, \tau) = \sum_{t=0}^{N} \sum_{q \in \mathcal{Q}_t} QA(\tilde{a}^* \mid q, \tau_t) \, \mathbb{I}[\tilde{a}^* \text{correct answer to q}]. \tag{1}$$

Algorithm Our algorithmic procedure is given in Algorithm 1. At the beginning of an episode, the QG takes the goal and returns a set of questions related to it. Then the QA module is applied at each step over the active set of questions $\mathcal{Q}_t$. When a question is answered correctly, the shaped reward

function returns a bonus $\lambda\, QA(\tilde{a}^* \mid q, \tau)$, where λ is a scaling factor, to the agent and the answered question is removed from the set of active questions.

Algorithm 1 Automatic auxiliary goal generation and reward shaping using EAGER

Input: θ_0 initial policy parameters, λ bonus scaling factor, ENV the environment and OPTIMISE an RL optimiser

```
for k=0,..., n_step do
    g, o_0, done_0 ←ENV.reset()
    Q_0 = {q_1, ..., q_k} ← QG_k(g)
    t ← 0
    while done_t not True do
        a_t ← π^{θ_0}(o_t)
        o_{t+1}, r_t, done_{t+1} ←ENV(a_t)
        r'_t, Q_{t+1} ← QA_SHAPE(Q_t, τ, r_t)
        if done_{t+1} is True then
            N ← t
            r'_N ← NEUTRALISE(r'_{1:N})
        end if
    end while
    Update θ_{k+1} ← OPTIMISE(r'_{1:N})
end for
```

```
function QA_SHAPE(Q_t, τ, r_t)
    for q in Q_t do
        ã* ← QA(q, τ)
        if ã* is correct answer to q then
            r'_t = r_t + λ QA(ã* | q, τ)
            Q = Q \ {q}
        end if
    end for
    return r'_t, Q
end function

function NEUTRALISE(r'_{1:N})
    r'_N ← r'_N − Σ_{t∈T_S} γ^{t−N} r'_t
    return r'_N
end function
```

Then we tune λ to ensure that no unsuccessful trajectories can get more reward than a successful one from the optimal policy π^*. The higher bound for the shaped reward of unsuccessful trajectories is $\lambda\, k$ and the lower bound for the reward of a successful trajectory is $\gamma^H\ r_H$, where k is the number of questions generated by the QG and H is the maximum length of an episode. Thus we obtain

$$\lambda < \frac{\gamma^H\ r_H}{k}. \tag{2}$$

Note that by making a less conservative hypothesis, i.e. assuming in the worst case the successful trajectory takes N<H steps, we could obtain a higher λ leading to faster learning [22].

4.2 A particular instance of the method in the BabyAI framework

We now explain how to adapt the EAGER method to train an RL agent in BabyAI, a language-conditioned environment where the agent has a limited number of steps to complete a language goal. In this environment the agent receives a reward if and only if it finishes the task successfully.

The BabyAI benchmark contains tasks with highly expressive language instructions e.g *Put the red box next to the green key*, ...). Thus they are rich enough to generate questions by masking words. In practice, we mask nouns and adjectives: this form of QG is very simple and can be automated using standard NLP techniques, thus it does not require expert knowledge. For instance, for the goal *Put the red ball next to the blue box*, using the token *«question»* as a mask we generate 4 questions: *Put the «question» ball next to the blue box*, *Put the red «question» next to the blue box*, *Put the red ball next to the «question» box*, *Put the red ball next to the blue «question»*.

The environments in our experiments are partially observable. Thus, our agent takes sequences of observations $(o_1, o_2, ..., o_t)$ as inputs of a recurrent network [13].

5 Experiments

5.1 Experimental settings

We use the BabyAI [7] platform to run our experiments. This platform relies on a gridworld environment (MiniGrid) to generate a set of complex instructions-following environments. It has been specifically designed for research on grounded language learning and related sample efficiency problems. The gridworld environment is populated with several entities: the agent, boxes, balls, doors, and keys of 6 different colors. These entities are placed in rooms of 8×8 tiles that are

connected by doors that could be locked or closed. The agent can do 6 primitive navigation actions such as `forward`, `toggle`, `pick up` to solve the language instruction (for instance `Pick up the red box`). It only has access to partial observations of its environment inside which irrelevant objects are randomly added. Moreover, the observations are in a symbolic space using a compact encoding, with 3 input values per grid cell, $8 \times 8 \times 3$ values in total. When the agent completes the task after N steps, it receives the reward $r_N = 1 - 0.9\frac{N}{H}$, where H is the maximum number of steps. During training, all rewards are scaled up by a factor of 20 to ensure a good propagation of the rewards. If the agent fails, the reward is 0. We focus our tests on tasks of varying complexity: PutNextTo, Unlock and Sequence. The task can also take place in one room `local` or two rooms `Medium`.

To train the QA module through supervised learning, we build a dataset of example trajectories associated to language goals using a bot provided in BabyAI, then we generate related questions and answers. To obtain a QA that can operate on various tasks, we use a mix of the PutNextTo, PickUp, Open, and Sequence tasks for generating training trajectories. This dataset is only used to train the QA, not to bootstrap policy learning. During training, we give as input of the QA the full trajectory — the list of observations and actions — and all the questions generated by the QG and we use the cross-entropy loss over its output distribution to update it. To train the QA to answer: "no_answer" and prevent it from guessing the answer by chance, we randomly associate some trajectories and questions from unrelated goals. For instance, we associate the trajectory from the goal: *take the red box next to the blue ball* and associate it with the question: *take the red «question» next to the blue key* generated from the goal *take the red box next to the blue key*. The QA must learn to pay attention to all the details of the question. It has to see that the trajectory describes an agent taking the *red box* but placing it next to an object other than the one in the question. Thus the good answer to the question is *no_answer* and not *box*.

Moreover, we empirically show in Section 5.4 that the QA is more efficient when it learns from a broad distribution of trajectories for similar tasks. The intuition behind such behaviour is that a QA trained on a narrow distribution of successful bot-generated trajectories will not recognise the noisy trajectories of the agent when it starts training. Thus the QA will too often answer "no_answer", resulting in no intrinsic reward and hurting the reward shaping efficiency. To produce a wider distribution using the given procedural bot, we force the bot to take with a certain probability a random action at each time step and only keep successful noisy trajectories as training examples. More details on the QA pre-training are given in Appendix B. All the tasks we use for training the QA and the agent are summarised with examples in Appendix A.

To evaluate our reward shaping framework, we use the Proximal Policy Optimization (**PPO**) algorithm [29], but our reward shaping method is algorithm agnostic. We compare our framework to PPO, ELLA and RIDE. RIDE [27] is an exploration method that does not use language and addresses sparse reward problem by rewarding impactful change in state.

We use Nvidia Tesla V100 with 10 cores to train our model and we use 4 seeds in each experiment.

Figure 2 presents learning curves for EAGER, ELLA and RIDE across 4 environments.

5.2 How does EAGER performs when sparsity increase?

In the PutNextTo and Unlock tasks, EAGER obtains results better or on par with ELLA (SOTA in BabyAI) without using expert knowledge. It performs also significantly better than RIDE for the tasks PutNext and Sequence and slightly better for Unlock.

For Unlock-Medium, EAGER overcomes a bottleneck. The general goal being *Open the «colour» door*, the agent has to first pick up the key of the corresponding *colour* before reaching the door to open it. ELLA rewards picking up keys, via the PICK low-level instruction chosen via expert knowledge. EAGER reaches comparable performances without the need for expert knowledge. Moreover, although this is not its main purpose, EAGER gets a better or close sample efficiency for most tasks.

5.3 How does EAGER perform with a sequence of tasks under a temporal constraint

The Sequence task adds a temporal constraint by chaining two tasks using 'before' or 'after' together with a high number of instructions (over 1.5M instructions in comparison with PutNext-Medium with 1440 instructions). Moreover both EAGER and ELLA decompose the goal into auxiliary objectives.

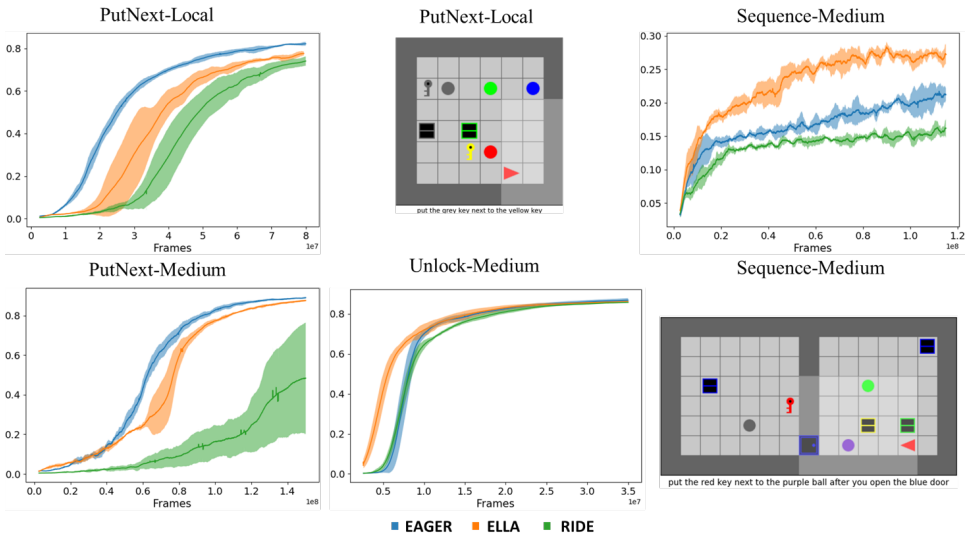

Figure 2: Average reward for EAGER and baselines for four tasks, with error regions to indicate standard deviation over four random seeds. For the tasks PutNext-Local and Sequence-Medium we give an examples of possible tasks with environment at time step 0.

This decomposition does not retain the temporal constraint, there is no notion of doing one auxiliary objective before another.

Our tests show that EAGER retains strong performance doing better than RIDE. It seems that it did not reach the performance of ELLA only due to the allocated time budget. Slower progress with EAGER can be attributed to the time when the agent is not good enough to efficiently trigger an intrinsic reward signal from the QA module. Indeed, at the beginning, trajectories are noisy and it is more difficult for the QA to exploit a trajectory with more rooms. As a result, it over-responds "no_answer" leading to a lesser intrinsic reward.

5.4 Is EAGER robust to QA performance?

At first glance, the reliability of the QA looks crucial to our method. However, the QA could be difficult to train in some environments e.g. if you want the QA to learn to answer in a large set of answers from a small number of example trajectories. This is why we tested the robustness of our method using the PutNextTo-Local task with two metrics: the success rate SR of the QA after pre-training and the distribution of example trajectories. For the former, we take the same QA at different training epochs and we determine its SR over a test dataset, then we train the agent using the reward shaping provided by this QA. Figure 3(left) shows the robustness of our method with agents that display similar training curves as soon as $SR > 0.56$. In Figure 3(right), we plot the SR of the QA when training the agent. Initially, the QA with a $SR \leq 0.56$ at pre-training time tends to have a higher SR. Indeed, the distribution of answers is less peaked and answers are often correct by chance. On the opposite, the QA with a $SR > 0.56$ answers "no_answer" and obtains no reward. However, in this case, the agent learns faster because it only receives a reward for meaningful answers. For the QA with a $SR \leq 0.56$, the SR along training first grows then decreases. First, the agent is biased by the intrinsic reward to follow a path that improves the SR, but once the agent learns to complete the trajectory leading to the extrinsic reward, the SR converges to the pre-training SR.

As explained in Section 5.1, we added noise to trajectories generated by the bot to compensate for a too narrow trajectory distribution. Figure 4 shows training curves for two environments for QA trained on wide trajectory distribution (WD) and narrow distribution (ND). The reward shaping method trained on (WD) learns faster because they efficiently reward the agent early in training.

5.5 How design choices on the QA module affect EAGER's performances ?

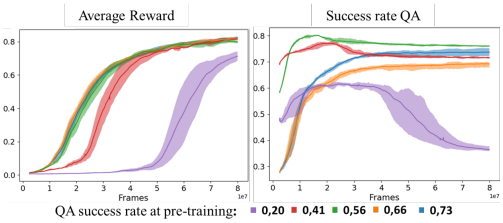

Figure 3: Average reward (left) and success rate of the QA (right) for the PutNextTo-Local tasks. The agent are trained with QA having different success rate after the pre-training.

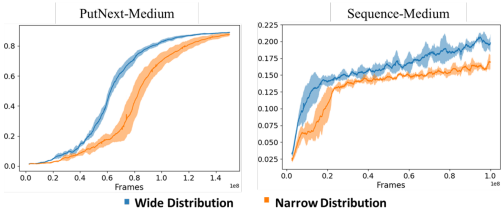

Figure 4: Average reward in tasks PutNextTo and Sequence for two distribution of bot trajectories used to train QA. A narrow and a wide distribution where noise is added to bot trajectories

In Section 4.1 we make two choices for the QA module and the associated intrinsic reward: first we added a "no_answer" response, second we rewarded each answer by the confidence the agent had in its own answer. To verify the influence of these choices over EAGER's performance, we use the PutNextTo task to compare EAGER against "EAGER \no_answer", "EAGER Simple", and "EAGER Simple \no_answer". "Simple" means that the agent received a binary intrinsic reward (1 for a good answer, 0 otherwise) and "\no_answer" means we suppress the "no_answer" solution. Figure 5 gives the results of these ablations. We can see that both the use of "no_answer" and the non-binary reward independently boost sample efficiency.

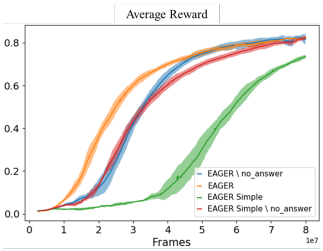

Figure 5: Average reward in PutNextTo with various ablations of the QA module.

6 Conclusion

In this work, we have proposed to leverage the abstraction and generalisation properties of language to build an automatic reward shaping method in the context of long horizon and sparse reward tasks. Our learning agent generates its own questions from the goal and rewards itself for correctly answering them, resulting in an efficient curriculum over auxiliary objectives. This is to be contrasted with ELLA [22] where expert knowledge is required for choosing auxiliary objectives. Besides, we do not call upon an oracle for getting linguistic description of environment states as in [24].

Limitations and Future Work EAGER assumes the QA system was pre-trained using a pre-existing set of example trajectories. Next steps will consist in investigating how to remove this limitation, e.g. by implementing autotelic strategies based on QG/QA learned online.

Besides, in this work we tested our method on BabyAI, a 2D environment with synthetic language. In the future, we would like to consider a more complex language, generating more complex questions than the one obtained by masking, and testing our method on more realistic environments with true human instructions, as in the ALFRED dataset [31].

References

[1] Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. *arXiv preprint arXiv:2006.07185*, 2020.

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Henge. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[3] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning (ICML)*, 2017.

[4] Arthur Aubret, Laetitia Matignon, and Salima Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*, 2019.

[5] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. In *arXiv preprint arXiv:1806.01946*, 2018.

[6] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[7] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[8] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

[9] Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 2022.

[10] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity-driven exploration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[11] Alison Gopnik and Andrew Meltzoff. The development of categorization in the second year and its relation to other cognitive and linguistic developments. In *Child development*, 1987.

[12] Goyal, Niekum, and Mooney. Using natural language for reward shaping in reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[13] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *arXiv preprint arXiv:1507.06527*, 2015.

[14] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, and Denis Teplyashin. Grounded language learning in a simulated 3d world. In *arXiv preprint arXiv:1706.06551*, 2017.

[15] Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. In *arXiv preprint arXiv:1906.00744*, 2019.

[16] U. Jain, S. Lazebnik, and A. G. Schwing. Two can play this game: Visual dialog with discriminative question generation and answering. In *Proc. CVPR*, 2018.

[17] Y. Jiang, S. S. Gu, K. P. Murphy, and C. Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[18] Iou-Jen Liu, Xingdi Yuan, Marc-Alexandre Côté, Pierre-Yves Oudeyer, and Alexander G. Schwing. Asking for knowledge: Training rl agents to query external knowledge using language. In *International Conference on Machine Learning (ICML)*, 2022.

[19] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.

[20] C. Lynch and P. Sermanet. Grounding language in play. In *arXiv preprint arXiv:2005.07648*, 2020.

[21] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016.

[22] Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. Ella: Exploration through learned language abstraction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[23] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *arXiv preprint arXiv:1704.08795v2*, 2017.

[24] Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette. Improving intrinsic exploration with language abstractions. In *arXiv preprint arXiv:2202.08938v1*, 2022.

[25] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML), volume 99, pages 278–287*, 1999.

[26] Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *arXiv preprint arXiv:2105.06453v2*, 2021.

[27] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations (ICLR)*, 2020.

[28] Clement Rebuffel, Thomas Scialom, Laure Soulier, Benjamin Piwowarski, Sylvain Lamprier, Jacopo Staiano, Geoffrey Scoutheeten, and Patrick Gallinari. Data-QuestEval: A referenceless metric for data-to-text semantic evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8029–8036, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.633. URL `https://aclanthology.org/2021.emnlp-main.633`.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[30] Thomas Scialom, Paul-Alexis Dray, Patrick Gallinari, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, and Alex Wang. Questeval: Summarization asks for fact-based evaluation. In *arXiv preprint arXiv:2103.12693v2*, 2021.

[31] Mohit Shridhar, , , , , Jesse Thomason, , , , , Yonatan Bisk Daniel Gordon, , , , Winson Han, , , Roozbeh Mottaghi, , Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *arXiv:1912.01734v2*, 2019.

[32] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. arXiv:1912.01734v2, 2018.

[33] Allison C Tam, Neil C Rabinowitz, Andrew K Lampinen, Nicholas A Roy, Stephanie CY Chan, DJ Strouse, Jane X Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *arXiv preprint arXiv:2204.05080*, 2022.

[34] Nicholas Waytowich, Sean L. Barton, Vernon Lawhern, and Garrett Warnell. A narration-based reward shaping approach using grounded natural language commands. In *International Conference on Machine Learning (ICML)*, 2019.

[35] Yunqiu Xu, Meng Fang, Ling Chen, Yali Du, Joey Tianyi Zhou, and Chengqi Zhang. Perceiving the world: Question-guided reinforcement learning for text-based games. In *arXiv preprint arXiv:2204.09597v2*, 2022.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.

- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the Method and Experiments Section 4 and 5.
   (b) Did you describe the limitations of your work? [Yes] See the Limitations paragraph of in Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We do not foresee major negative societal impacts of this work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We do believe our work conforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We give the URL in the introduction.
   (b) Did you specify all training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the Section 5 and the Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran our main experiments with multiple random seeds and plotted curves with standard deviations.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the end of section 5.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use open-source environments, cited in the code and in the paper text.
   (b) Did you mention the license of the assets? [Yes] We inherit the license in our attached code repository.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include the code of our approach
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]