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

Thomas Carta
Inria - Flowers team
Université de Bordeaux
thomas.carta@inria.fr

Sylvain Lamprier
ISIR Sorbonne Université
Univ Angers, LERIA,
SFR MATHSTIC, F-49000 Angers, France
sylvain.lamprier@lip6.fr

Pierre-Yves Oudeyer
Inria - Flowers team
Université de Bordeaux
Microsoft Research Montreal
pierre-yves.oudeyer@inria.fr

Olivier Sigaud
ISIR Sorbonne Université, Paris, France
olivier.sigaud@isir.upmc.fr

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 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 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 [12]. 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, 25, 13, 41]. The domain can be subdivided into language-conditioned RL (LC-RL), in which language conditions the formulation of the problem[2, 13], and language-assisted RL, where language helps the agent to learn [16, 11, 1, 10]. 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

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

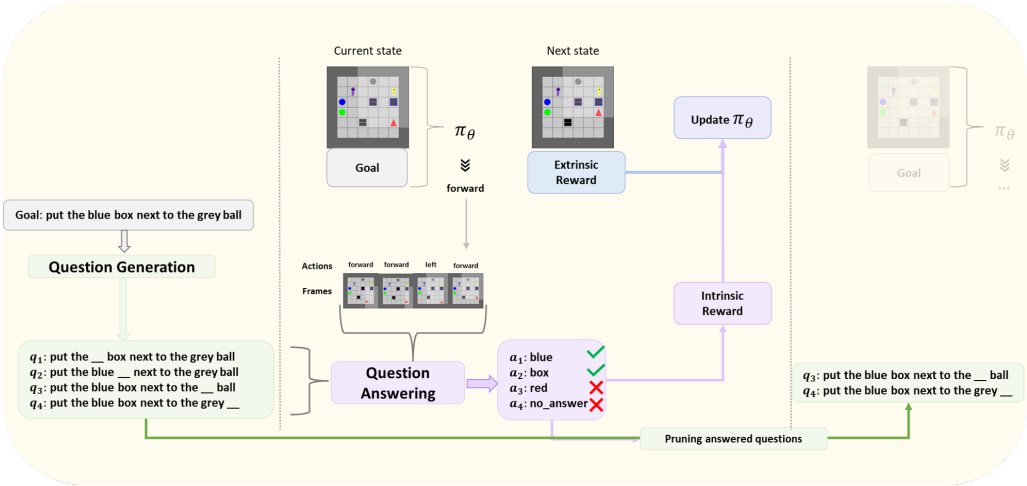

Figure 1: During training, the agent uses the goal to generate relevant questions using its question-generation module **QG**. Then, it attempts at answering them from current trajectories at each step with its question-answering module **QA**, by looking at the trajectory. When it succeeds, it obtains 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.

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, 18]. 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 [26].

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 [36], [34], 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 QA module is trained using a set of successful example trajectories. If the agent follows a path too different from correct ones at some point in its trajectory, the QA module

[1]Exploit question-Answering Grounding for effective Exploration in language-conditioned Reinforcement learning, see `https://github.com/flowersteam/EAGER` for access to the code.

will not answer the question correctly, resulting in zero intrinsic reward. 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.

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 [8]: 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. Furthermore, although we use example trajectories to train the QA, their use is much more parsimonious than training an agent using behavioural cloning [8], as we show in Appendix C.

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 [27, 8, 23, 24]. We reuse the BabyAI platform [8], 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 [9]. Here, we consider *instruction following* agents which receive external instructions and rewards [15, 5, 18].

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 [18]. Another one explicitly decomposes high-level tasks into low-level ones as in policy sketches [3]. ELLA [26] 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 [13] proposes to map intermediate natural language instruction to intermediate rewards. Similarly, [40] 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 [26] 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 [28, 39]. This extends classical approaches to intrinsically motivated exploration [4], such as count-based methods [7] or RIDE [32]. 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 [11], 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 [41], questions are used to identify sub-tasks and prune irrelevant actions. In AFK [22], 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 [17]. 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 [36, 34].

3 Problem statement

We set ourselves in the standard framework of LC-RL with an augmented Partial Observation Markov Decision Process [38] $\mathcal{M}$ defined by the tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{Z}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{G}$, $\mathcal{V}$, $\mathcal{I}$, $\mathcal{R}$, γ), with $\mathcal{S}$ the state space, $\mathcal{Z}$ the observation space, $\mathcal{A}$ the actions space, $\mathcal{G}$ the goal space, $\mathcal{V}$ the vocabulary of goal instructions. $\mathcal{O}$ stands as the observation function $\mathcal{O} : \mathcal{S} \rightarrow \mathcal{Z}$, which maps states to the observations space. $\mathcal{I}$ is the instruction function $\mathcal{I} : \mathcal{G} \rightarrow \mathcal{V}^{isize}$, which maps any goal in $\mathcal{G}$ to the set of language instructions, which correspond to sequences of $isize$ symbols (the empty symbol ϵ belongs to $\mathcal{V}$ to allow variable instruction sizes). $\mathcal{R}$ is a goal-conditioned state action reward function, with $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow [0, 1]$ the extrinsic reward received for some goal $g \in \mathcal{G}$ and γ the discount factor. For simplicity, we note in the following $r_t^g = \mathcal{R}(s_t, a_t, g)$ as the reward obtained at step t of any episode with goal g.

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 $g \in \mathcal{G}$, expressed by $\omega^g = \mathcal{I}(g)$. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$ is the transition function. Using RL, we search for an optimal goal-conditioned policy π^*, such that $\pi^* : \mathcal{S} \times \mathcal{V}^{isize} \rightarrow \mathcal{A}$ maximises the discounted expected return $R_t^{\pi_g} = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k \, r_{t+k+1}^g]$. We consider in this work sparse reward problems, where $r(s, a, g)$ returns 1 for any state s such that $d(s, g) \leq \epsilon$, for a given distance function d and a specified threshold ϵ, and 0 otherwise. Moreover, we assume a limited number of steps at most H steps. These two conditions result in a hard exploration problem.

To deal with those problems, many methods aim at densifying rewards, by focusing on auxiliary objectives during training, whose accomplishment can help the agent to reach the goal g at hand. In previous work, such as ELLA [26], the selection of relevant objectives required the intervention of an expert (in the form of expert annotations and example trajectories), which can be problematic because new expert trade-offs have to be established for each new environment. Thus, an automated way must be found to recover the relevant auxiliary objectives, measure their completion, and associate the appropriate reward.

Rather than relying on expert knowledge for defining auxiliary objectives, we assume that we have access to a set of trajectories of successful examples coupled to their respective instructions $\{(\tau_0, \omega_0^g), ..., (\tau_n, \omega_n^g)\}$, where $\tau_n = (o_i, a_i)_{i \in [|0,k|]}$ with k the number of steps. For any goal instruction ω^g, we consider a function f that aims at generating a set of auxiliary goals of g, such that all $g' \in f(\omega^g)$ belong to $\mathcal{G}^g$, with $\mathcal{G}^g \subset \mathcal{G}$ the set of goals that help training the agent towards g. Then, for a trajectory τ and any $g' \in f(\omega^g)$, a function h determines the probability $h(\tau, g') \in [0, 1]$ that the auxiliary objective g' has been achieved. We train h using the example demonstrations such that we can leverage its generalisation abilities to use h for unseen trajectories.

Please note that Behavioural Cloning methods also rely on the exploitation of expert trajectories, by learning the policy π_{BC} that maximises the log-likelihood $\mathcal{L}_{BC}$:

$$\mathcal{L}_{BC} = \sum_{i=0}^{n} \log P(\tau_i) \text{ where } \log P(\tau) = \sum_{j=0}^{k} \log(\pi_{BC}(a_k|o_k)\mathcal{T}(s_{k+1}|s_k, a_k)).$$

However, the latter uses demonstrations to train an agent to copy an expert based on a data set of example trajectories. This technique is only effective when the agent is close enough to the demonstrated examples. A large number of demonstrations are therefore required to cover the space of states that the agent may encounter. Methods such as GAIL [19] can partially circumvent this problem by forcing the agent to stay on known trajectories, but this impairs generalisation. Our method is much more parsimonious in the use of such examples: because it generalises well, the

agent can receive intrinsic rewards even in areas not encountered in the expert demonstrations (see Appendix C).

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. .These two modules fulfil the role of the functions f and h defined in Section 3. EAGER is inspired from works like QuestEval [36] and Data-QuestEval [34], 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 method 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 the goal linguistic instruction ω^g of $g \in \mathcal{G}$ is a highly expressive language instruction (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 $QG_k(\omega^g)$ that can be seen as auxiliary objectives. 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. Thus, $\tilde{A}$ contains (q, a^*) pairs of questions and expected answers. The QA module returns the probability for all $\tilde{a} \in \tilde{A}$, that $\tilde{a}$ answers question q, given a trajectory τ_t, where $\tau_t = (o_i, a_i)_{i \in [|0,t|]}$ are state-action pairs and t the time step. We note $\tilde{a}^* = QA(q, \tau_t)$ the answer greedily generated from the module and $QA(\tilde{a}^* \mid q, \tau_t)$ 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 full 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 $\mathcal{Q}_0 = QG_k(\omega^g)$ 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 [30]. 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 at time step t: $QA(\tilde{a}^* \mid q, \tau_t)$. 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. Using intrinsic rewards, we modify the global reward from r to r' through a policy invariant transformation [29], which ensure that the new policy π^* is optimal for both $\mathcal{M}$ and $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{G}', \gamma)$.To do so, we ensure that only successful trajectories get the same return with or without the reward shaping by substracting 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. [26] 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 substracted at the last step and the agent improves using only the extrinsic reward. Further details are provided in the appendix.

The QA system is pre-trained with successful trajectories, which prevents reward hacking. Indeed, if the agent individually completes 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 $\lambda \, QA(\tilde{a}^* \mid q, \tau_t)$, where λ is a scaling factor, to the agent and the answered question is removed from the set of active questions.

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 [26].

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**
  $\omega^g, o_0, \text{done}_0 \leftarrow \text{ENV}.reset()$
  $\mathcal{Q}_0 = \{q_1, ..., q_k\} \leftarrow QG_k(\omega^g)$
  $t \leftarrow 0$
  while done_t not True **do**
    $a_t \leftarrow \pi^{\theta_0}(o_t)$
    $o_{t+1}, r_t, \text{done}_{t+1} \leftarrow \text{ENV}(a_t)$
    $r'_t, \mathcal{Q}_{t+1} \leftarrow \text{QA_SHAPE}(\mathcal{Q}_t, \tau_t, r_t)$
    if done_{t+1} is True **then**
      $N \leftarrow t$
      $r'_N \leftarrow \text{NEUTRALISE}(r'_{1:N})$
    end if
  end while
  Update $\theta_{k+1} \leftarrow \text{OPTIMISE}(r'_{1:N})$
end for

function QA_SHAPE($\mathcal{Q}_t, \tau_t, r_t$)
  for q in $\mathcal{Q}_t$ **do**
    $\tilde{a}^* \leftarrow QA(q, \tau_t)$
    if $\tilde{a}^*$ is correct answer to q **then**
      $r'_t = r_t + \lambda\, QA(\tilde{a}^* \mid q, \tau_t)$
      $\mathcal{Q} = \mathcal{Q} \setminus \{q\}$
    end if
  end for
  return $r'_t, \mathcal{Q}$
end function

function NEUTRALISE($r'_{1:N}$)
  $r'_N \leftarrow r'_N - \sum_{t \in T_{\mathbb{S}}} \gamma^{t-N} r'_t$
  return r'_N
end function

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 among which *Put the «question» ball next to the blue box*.

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 [14].

5 Experiments

5.1 Experimental settings

We use the BabyAI [8] 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,[2] $8 \times 8 \times 3$ 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

[2] 3 integers: one representing the shape of the object, one its color, and one its state. For instance, (4, 1, 1) represents a closed green door

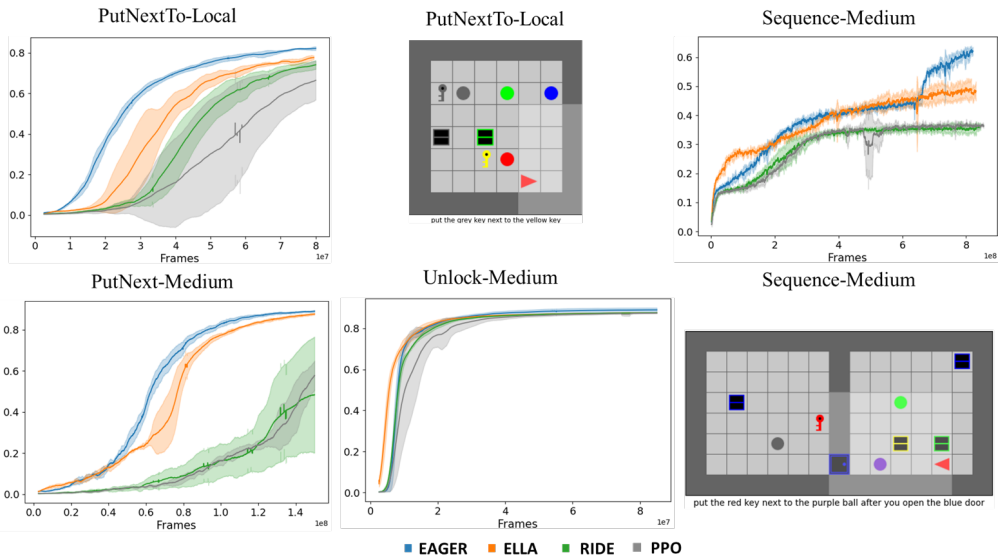

Figure 2: Average reward for EAGER and baselines for four tasks, with error regions to indicate standard deviation over four random seeds. For the PutNext-Local and Sequence-Medium tasks, we give an example of possible tasks with environment at time step 0.

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.1. 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 [35], but our reward shaping method is algorithm agnostic. We compare our framework to PPO without reward shaping, ELLA and RIDE. RIDE [32] 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. Table 1 describes the assumptions and the type of expert knowledge required by the three reward shaping methods. It clearly appears that EAGER requires less expert human intervention than ELLA.

Table 1: All the assumptions and expert knowledge required for RIDE, EAGER, and ELLA

Method	**Number of expert demonstration per task**	**Human Expert Intervention**	**Automated parts done by the agent**
RIDE	0	None	Determining if a new state is impacting
EAGER	7.500 noisy bot trajectories (see supplementary *Wide distribution of trajectories*)	Determining what words are nouns or adjectives	Determining relevant auxiliary objectives Determining auxiliary objectives completion Determining auxiliary objectives associated reward
ELLA	15.000 bot trajectories	Determining the class of relevant auxiliary objectives Determining auxiliary objectives associated reward	Determining relevant auxiliary objectives among the predetermine class Determining auxiliary objectives completion

5.2 How does EAGER perform when sparsity increases?

In the PutNextTo and Unlock tasks, EAGER obtains results better than ELLA (SOTA in BabyAI) without using expert knowledge. It also performs significantly better than RIDE for the tasks PutNext and Sequence and slightly better for Unlock. The better performance of EAGER with respect to RIDE is not surprising as the EAGER agent receives some indications based on example trajectories through the QA module.

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 better performance (see the statistical test in Appendix C) without the need for expert knowledge. Moreover, although this is not its main purpose, EAGER gets a similar or better 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. 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 and ELLA. The slow progress of EAGER at the beginning 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

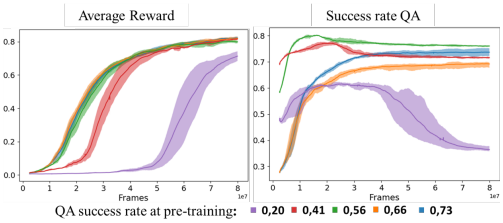

Figure 3: Average reward (left) and success rate of the QA (right) for the PutNextTo-Local tasks. The agent are trained with QA having different success rate after pre-training.

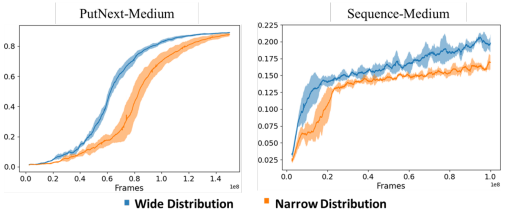

Figure 4: Average reward in tasks PutNextTo and Sequence for two distributions of bot trajectories used to train QA. A narrow and a wide distribution where noise is added to bot trajectories

the SR, but once the agent learns to complete the trajectory leading to the extrinsic reward, the SR converges to 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 do design choices on the QA module affect EAGER's performance?

In Section 4.1 we made 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 suppressed 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.

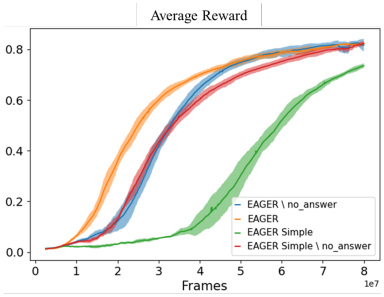

Figure 5: Average reward in PutNextTo with various ablations of the QA module.

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 [26] 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 [28].

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 [37].

Acknowledgments and Disclosure of Funding

This work benefited from the use of the Jean Zay supercomputer associated with the Genci grant A0091011996, as well as from the ANR DeepCuriosity AI chair project.

This work has been supported by the Horizon 2020 PILLAR-robots project (grant number 101070381)

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] Dzmitry Bahdanau, Felix Hill, Edward Hughes Jan Leike, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. In *arXiv preprint arXiv:1806.01946v4*, 2019.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Stefano Ermon Jonathan Ho. Generative adversarial imitation learning. In *arXiv preprint arXiv:1606.03476v1*, 2016.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *arXiv preprint arXiv:1412.6980v9*, 2014.

[21] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *arXiv preprint arXiv:2110.06169v1*, 2021.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] 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.

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

[31] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Adaptive Computation and Machine Learning*, 2017.

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

[33] C. E. Rasmussen and C. K. I Williams. Gaussian processes for machine learning. In *Adaptive Computation and Machine Learning*, 2005.

[34] 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`.

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

[36] 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.

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

[38] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[39] 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.

[40] 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.

[41] 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

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]

This supplementary material provides additional results and discussion, as well as implementation details.

- Section A summarises the different tasks and the assumption used in RIDE, EAGER, ELLA.
- Section B gives more details about training of the QA module and the agent. It also includes explanations of how we built the training data set for the QA module.
- Section C gathers several results on EAGER: comparison with behavioural cloning, generalisation capacity of QA, robustness results of EAGER...
- Section D contains a commented version of the EAGER algorithm.
- Section E summarises hyperparameters.

A Tasks description and assumptions used for the different method of reward shaping

Table 2 describes the tasks used in the experiments with an example and if it has been used to train the QA module or the agent. The Unlock and Open tasks have the same type of instructions, the agent can nevertheless see the difference because in the Unlock task, the door is a solid square where in the Open task, the door is just a border.

Table 2: Tasks description, ✓means that the task is used for training the QA resp. the Agent

Task	Explanation	Example	Train QA	Train Agent
PutNextTo	put an object next to another	*put the purple ball next to the blue key*	✓	✓
PickUp	pick up an object	*pick up a red box*	✓	
Open	open a door (does not require a key)	*open the green door*	✓	
Unlock	open a door using the key of the same colour	*open the green door*		✓
Sequence	sequence of two of the previous tasks: task 1 before/after task 2	*put the blue key next to the red box before you open the grey door*	✓	✓

B Training details

In this section we explain how we create the QA training data set, the architecture of the agent used for RL training, and give more details on the baselines.

B.1 QA architecture and pre-training

QA architecture The QA architecture is based on the Episodic Transformer architecture [30] depicted in Figure 6. Using multimodal transformers, the QA can direct its attention over certain observations in the trajectory about the words used for the question and the previously taken action. Thus we can train the QA over the full trajectory and use it on partial trajectories (up to time step t) at test time.

QA training data set We train the QA with a mix of 4 tasks: Open-Large, PickUp-Large, PutNextTo-Local, and Sequence-Medium. We use a bot provided with the BabyAI platform that procedurally solves the environment. Using it, we generate 7500 example trajectories for each task. We use a mix of tasks to push the QA to leverage the compositionality of language. Indeed, the Sequence task is created by putting in sequence two tasks from Open, PickUp, and PutNextTo. Compared to QA trained on an individual task, our QA needs fewer examples per task. Thus the QA can use the environment with the goal *put the blue key next to the red box then open the grey door* to ground the instruction both for PutNextTo and Open.

Adding "no_answer" questions To train the QA to respond: "no_answer" and prevent it from hazard-guessing the answer, we randomly associate certain paths and questions from unrelated objectives. To generate these questions, for each new trajectory generated by the bot, we take a goal among the last three used. If this goal differs from the goal used for the trajectory, we use it to

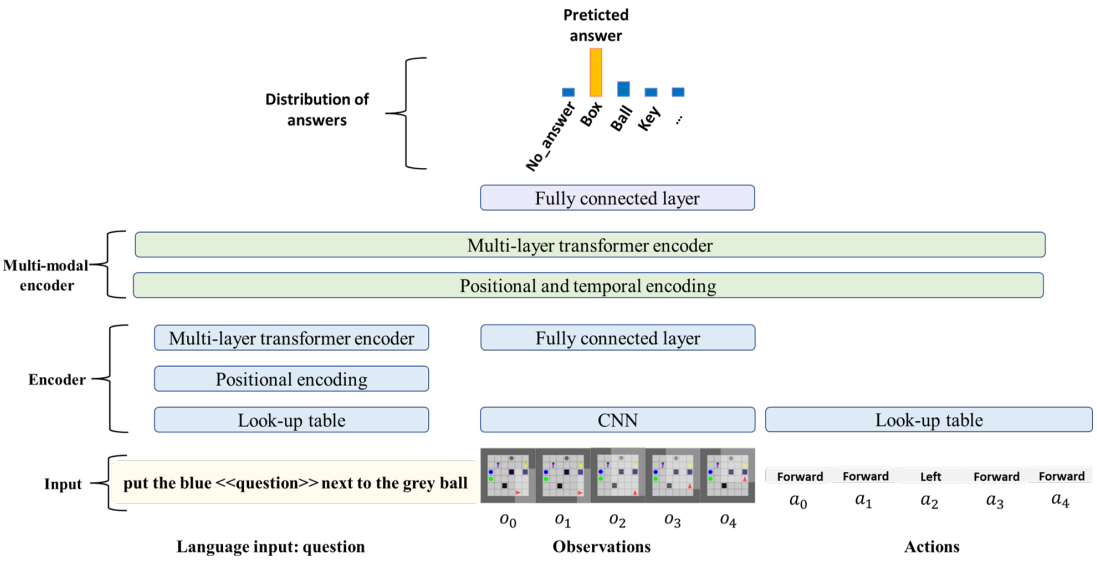

Figure 6: The QA architecture is based on the architecture of the Episodic Transformer [30]. To predict the correct answer the QA model received a natural language question, visual observations, and actions generated by the agent from the beginning of an episode (see Figure 8). Here we show an example that corresponds to the 5th time step of an episode. After processing the question with a transformer-based language encoder, embedding the observation with a convolutional neural network CNN and passing actions through a look-up table, the agent outputs the correct answer. During training, we use full trajectories. At test time, we ask all the questions in the active set of questions using the current trajectory.

create questions that are associated with "no_answer". To avoid biasing the dataset with too much of these questions, we randomly keep some of the generated "no_answer" questions. The probability to keep a question depends on the number of words in common between the goal corresponding to the trajectories and the random goal. If these two goals share a lot of words in common, it is harder for the QA to see that it cannot answer the question generated from the random goal, thus we want to favour these negative examples in our dataset. We define $\mathbb{P}(\text{keep question}) = \frac{0.325}{1+\exp(6.75-3 \text{ number words in common})} + 0.095$. The hyperparameters are chosen empirically to obtain a number of "no_answer" questions similar to the number of questions generated for other words.

Wide distribution of trajectories As explained in the experimental settings and empirically demonstrated in the experiments, a broad distribution of trajectories improves the QA and the efficiency of EAGER overall. To obtain such distribution from the procedural bot, we replace at each time step with a probability p the action of the bot by a random action in: *turn right*, *turn left*, *go forward*, *pick up*, *drop* as it is shown in Figure 7 . For each new example trajectory, we randomly select p, with probability 0.45 for $p = 0$ (no random action) and with a probability of $[0.35, 0.1, 0.1]$ for $p \in [0.1, 0.4, 0.8]$ respectively. We use such a distribution to sample p to ensure enough good quality trajectories in the dataset. If the bot cannot complete the task due to the added noise, we discard the trajectory from the set of example trajectories.

We train the QA using the cross entropy loss with a batch size of 10 and a learning rate of 10^{-4} using Adam [20]. We multiply the learning rate by 0.1 every 5 epochs. Table 3 summarises the hyperparameters used.

B.2 Agent architecture and training

We use the actor-critic architecture and the PPO implementation proposed in BabyAI [8] (Figure 8). This PPO implementation uses the default hyperparameters (see Table 4). The output of the actor is the distribution of actions. There are six possible actions: turn left, turn right, forward, pick up, drop, toggle and done (to signify the completion of the task). The output of the critic is the value of the current state.

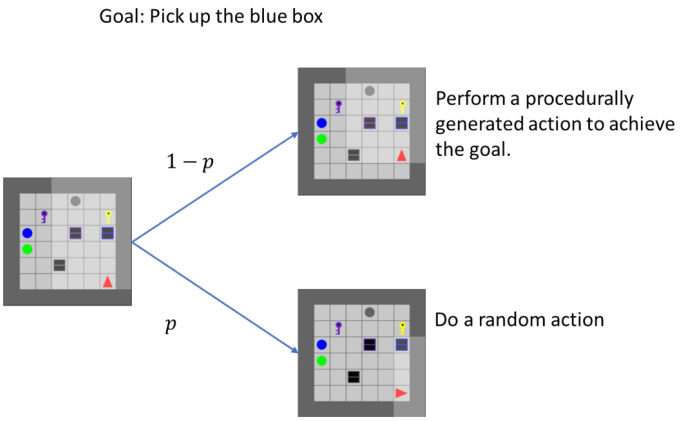

Figure 7: Choice of a new action during the generation of the wide distribution of trajectories. At each time step, the agent can perform a random action with probability p.

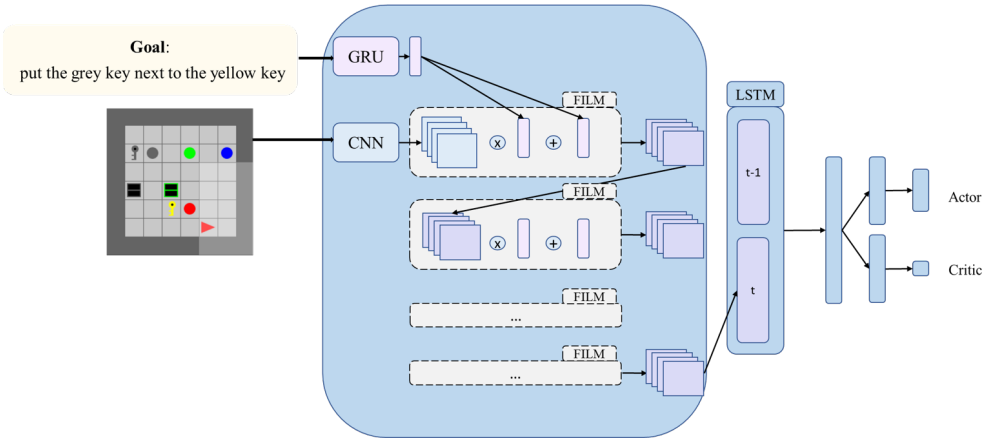

Figure 8: The actor-critic architecture uses a multimodal encoder that mixes language and image using a feature-wise affine transformation FiLM Perez et al. [31]. Then the encoded representation passes through an LSTM to process the history of observations. Finally, the output is used in an actor and critic heads.

B.3 RIDE

RIDE is a method that does not use language and gives an intrinsic reward encouraging agents to explore actions that significantly alter the state of the environment, as measured in a learned latent space. To use it as a baseline, we reimplemented the RIDE algorithm from the open-source implementation.[3]. We kept the same architecture as the original work and adapted RIDE to the on-policy setting of our PPO algorithm by updating the dynamics models once per batch of on-policy rollouts. We used the hyperparameters values published in the code repository for the coefficients on forward dynamics loss and inverse dynamics loss (10 and 0.1 respectively), as well as the published value for learning rate of 10^{-4}. We took the intrinsic reward coefficient $\lambda = 0.5$ as it was empirically shown in ELLA to be the best coefficient.

B.4 ELLA

ELLA is a reward shaping approach for guiding exploration based on the principle of abstraction in language decomposing high-level goals into low-level auxiliary objectives. Two classifiers are learned: a termination classifier that determines when a certain auxiliary objective has ended and a relevance classifier that determines which auxiliary objective is relevant for the high-level goal. For instance, *pick up yellow key* is relevant for the goal *open yellow door*. The termination classifier is trained from labelled trajectories and the relevance classifier is learned online. Expert knowledge is

[3] `https://github.com/facebookresearch/impact-driven-exploration`

required to label trajectories and to determine which set of auxiliary objectives is interesting such as: *go to object* or *pick up object*. For this baseline, we rerun the open-source code.[4]

B.5 EAGER and RIDE

EAGER and RIDE reward different aspects of the exploration, so their combination has the potential to outperform the two methods taken separately. In our experiments we simply add the intrinsic reward returned by EAGER and RIDE, weighted with a shape reward weight λ_{RIDE} and λ_{EAGER}. Nonetheless, we cannot use the technique explained in Section 4 to find the optimal value of λ. Indeed, this technique is based on the sparsity of the reward, and adding the intrinsic reward from RIDE invalidates this condition. Thus we have to resort to ad hoc methods for tuning λ_{RIDE} and λ_{EAGER} in the combined version.

We observe that the values of λ used by ELLA and EAGER are of the same order of magnitude, so we use the value found in Mirchandani et al. [26] for the combination of ELLA et RIDE. Thus we take $\lambda_{RIDE} = 0.05$ and $\lambda_{EAGER} = 0.1$, we test this combination on PutNext-local, PutNext-medium and Unlock-Medium tasks.

We obtain better results for the task Unlock-Medium Figure 9. For the PutNext-local and PutNext-medium tasks the combination of RIDE and EAGER performs worse. Thus this combination is highly sensitive to the value of λ_{RIDE} and λ_{EAGER}.

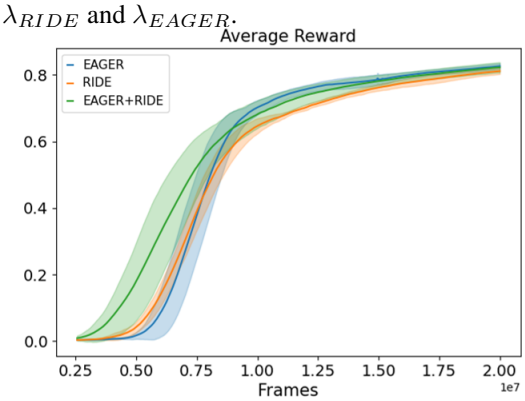

Figure 9: Average Reward for the Unlock-Medium task for EAGER, RIDE and the combination of EAGER and RIDE.

C More results on EAGER

Statistical significance of the results for Unlock-Medium For Unlock-Medium the last value for the average return of EAGER is 0.8903, higher than the one of ELLA (0.8765), RIDE (0.8761) and PPO without reshaping (0.8747), see Figure 10. The standard deviation of EAGER is higher (0.011) than the one of ELLA (6.10^{-4}), RIDE (2.10^{-4}) and PPO without reshaping (0.0021). Indeed, EAGER is still learning and exploring, whereas the baselines are stuck in a local minimum. Moreover, we use a Welch's t-test to test the null hypothesis: equality between the mean of EAGER and ELLA at the end of the curve. The test rejects this hypothesis with $p_{\text{value}} = 1.9\,10^{-31}$. Thus EAGER significantly outperforms ELLA in this task.

Comparison with behavioural cloning and offline RL To train the QA, we used a data set of 7.500 example trajectories per task. One can ask if these trajectories can be used to train an agent through behavioural cloning to obtain results similar to the ones obtained with EAGER. However, experiments performed in [8] show that tasks like PutNextTo-Local require at least 244.000 example trajectories to be learned successfully. Nonetheless, we trained an agent with behavioural cloning using our data set. For the PutNextTo-Local task, our trained agent failed to complete any goal. But when we examine generated trajectories, it seems to display relevant behaviour, going around the objects of interest. This behaviour is coherent with the results given in Mirchandani et al. [26] with 50.000 example trajectories needed for learning goals like *Go to red ball*. Thus EAGER requires much less demonstration trajectories than behavioural cloning, as it is more efficient to train a QA

[4] `https://github.com/Stanford-ILIAD/ELLA/tree/22884a3da33da2534754693280a47bb0d99eb8c5`

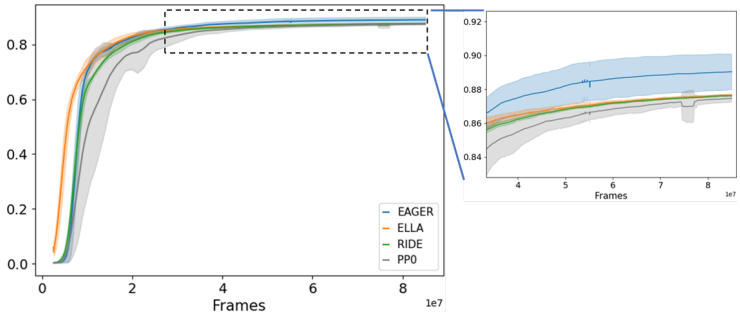

Figure 10: Average Reward for EAGER and baselines.

module than a BC policy. To further this analysis, we have implemented IQL, a recent offline RL algorithm [21]. IQL never needs to evaluate actions outside the data set, but still maintains some generalization capability. Nonetheless, this method also fails due to the small number of trajectories, with a success rate of 0 for PutNextTo-local tasks. To ensure that the implementation is correct, we tested IQL with a simple GoTo task. It obtains a non zero result on the test set with 6% of the test trajectories being successful. Thus the data set used for training the QA cannot be used to warm start an agent. This result is not too surprising, it is indeed easier to learn from a data set to recognise a pattern (as the QA module does) rather than learning a policy Bahdanau et al. [6].

Generalisation properties of the QA module We also verified that the QA can generalise by correctly answering questions formulated from goals never seen in training. To check this, we trained the QA with questions and trajectories generated from the Sequence-Medium task. To test generalisation, we only used in the test data set trajectories and questions generated from goals that are not present in the train data set. On the test set, we obtained a success rate of 0.67 to be compared with 0.74 when we tested with trajectories with already seen goals. Thus the QA module can generalise to new goals, making EAGER efficient even for goals not present in the QA training data set.

Ablation of the policy invariance To ensure the policy invariance requirement (see Section 4.1) we subtracted the shaped reward at the final step of successful trajectories by neutralizing the intrinsic reward. In Figure 11, the curve in orange is the learning curve when we do not apply neutralization. The agent learns faster that the one on which neutralization is applied because it receives more indication through reward. However, it ends up stuck in a local minimum because the final policy is influenced by the intrinsic reward.

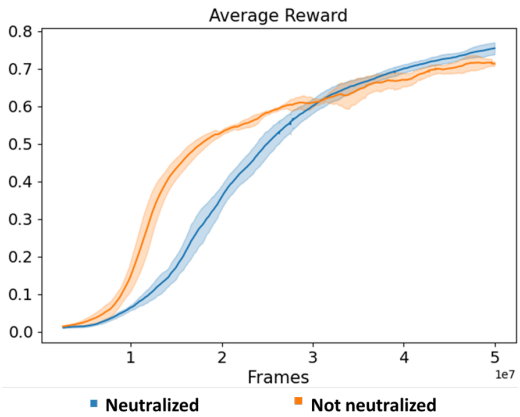

Figure 11: Ablation of the neutralization of the intrisic reward on the task PutNextTo-local.

QA performance along the trajectory In order to minimise human intervention, we only use whole trajectories to train the QA. It is legitimate to ask whether the performance of the QA module changes along the trajectory. To determine this, we used the number of different attempts before

successfully answering a question as a proxy to measure the performance of the QA module on short trajectories.

We used the PutNext-local task which has the following structure: "Put the adjective_1 noun_1 next to the adjective_2 noun_2". Measuring the number of different attempts before finding the correct answer, we began to count after the first answer different from "no_answer" (before that, the QA estimates that it cannot answer). For this task, after 100 trajectories, we found:

Token	Average number of attempts
adjective_1	4.37
noun_1	2.55
adjective_2	1.69
noun_2	1.25

It appears that the QA module needs on average twice as many attempts to guess the answer at the beginning (when the trajectory is short and partial) rather than at the end of the trajectory. Nonetheless, in practice, this does not seems to impact the performance of the methods, which underlines the robustness of EAGER.

Results for the task Sequence with an extended time budget In order to understand why EAGER suddenly does better than ELLA after 6.5e8 frames Figure 12, we empirically observed which tasks were successful before and after this threshold. The goals in Sequence task are a combination of two tasks among 'Pickup', 'Go to','Open', and 'Put Next to'. Nonetheless, the later is already a combination of the tasks 'Pickup' and 'Go to', thus more difficult than any of them alone. In Figure 13 we observe the reward obtains by ELLA and EAGER for goals without 'Put Next to' tasks in it, with 1 'Put Next to' and 2 'Put Next to'. The former type represents around 50% of the goals, goals with 1 'Put Next to' represents 40% and the latter accounts for 10% of the goals. Thus, after 6.5e8 frames, the improvement of the EAGER agent for the goals with 1 and 2 'Put Next to' allows for an important increased in the total reward.

Up to 6.5e8 frames EAGER and ELLA only mostly succeed on Sequence tasks that does not contains a 'Put Next to' goal. However, when the agent starts to have a trajectory that brings it closer to success for tasks with 'Put Next to' in them, it seems that EAGER recognises it and rewards it. In contrast ELLA is much slower as it must first learn to break down complex goals (with one or two 'Put Next to' tasks in them) into auxiliary objectives (using its relevance classifier) before it can use them for reward shaping.

If we look at the curves of ELLA and EAGER we can try to interpret all the different phases:

- At the beginning (0 to $1,5.10^8$ frames) ELLA does better than EAGER because the relevance classifier breaks down the goal into auxiliary objectives quite easily, whereas EAGER is hampered by the fact that the trajectories are shorter and performs poorly (as shown in the paragraph: *QA performance along the trajectory*)
- After ($1,5.10^8$ to $3,5.10^8$) the agent's trajectories improve and become longer, EAGER gains in performance and overtakes ELLA
- From $3,5.10^8$ to $6,5.10^8$ EAGER and ELLA manage to achieve almost the same goals (goals with no 'Put Next to tasks in it)
- From $6,5.10^8$ frames the agent's trajectories for complex goals are longer, EAGER is performing well while ELLA is training the relevance classifier to decompose complex goals into auxiliary objectives (this is learned online from the agent's successful trajectories which explains the slow performance)

Verifying that the QA does not guess With the QA, the agent can self-check the followed instructions. Due to the format of the generated questions (i.e. masking a word) and proceeding by elimination depending on the object present in the environment, a simple QA could guess the answer, breaking the EAGER method. Thus, to avoid such issues we added so-called "no_answer questions" to the QA training data set. These questions are examples where the trajectory does not correspond to the question, e.g. a trajectory corresponding to the goal "pick up the red box then pick up the red

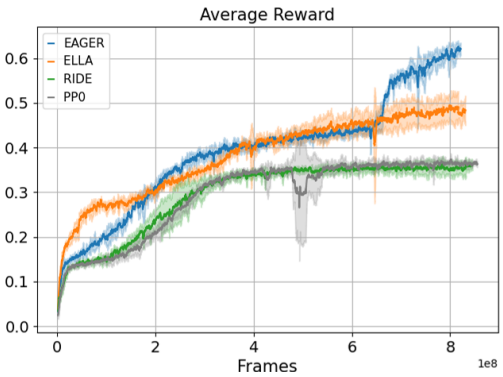

Figure 12: Average reward for the task Sequence.

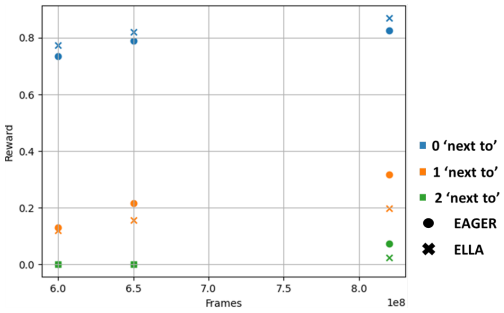

Figure 13: Average reward over 500 goals and 4 seeds, in Sequence, distributed in three categories: goals without 'Put next to' task in it and goals with 1 or 2 'Put next to' tasks.

ball" is associated to the question "pick up the «question» key then pick up the red ball". The QA must learn to associate each element to see that the path is not associated to the correct question and appropriately answer "no_answer". In practice, this prevents the QA from guessing the answer, it needs to wait for the agent's path to match before answering. Thus for a question such as "pick up the «question» box then pick up the red ball", even if the only box in the room is red, the QA must answer "no_answer" as long as the agent has not completed the corresponding auxiliary objective.

We verified that the QA correctly acts by checking a hundred trajectories with the possibility to guess an answer based on the linguistic input, such as with the goal "pick up the red ball then pick up the red ball". We give 4 examples among the tasks we have checked in Figure 14. We did not find any instance where the QA guessed the answer by chance right from the beginning of a trajectory. Indeed, even if it is possible to just use linguistic elements to answer a question such as "pick up the «question» ball then pick up the red ball", the QA cannot associate the answer "red" to any element of the trajectory and thus returns "no_answer".

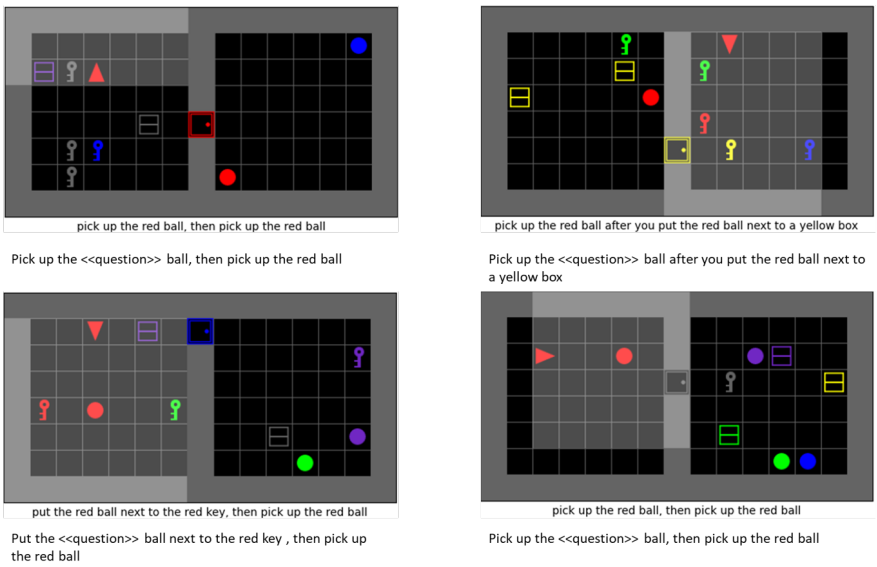

Figure 14: Four examples and possible question at time step $t = 0$ where the QA could guess the answer either linguistically or by elimination process. In all of these examples the QA returns no_answer because the trajectory does not correspond to a possible answer

Relation between sample efficiency and success rate of the QA In the experiments on EAGER robustness, we looked at the robustness of EAGER relatively to the success rate of the QA during pre-training. However, we only used some values of SR — by sampling the QA at different training

epochs— to train the RL agent. To strengthen our point, we used a Gaussian Process model Rasmussen and Williams [33] to fit the relation between the success rate of the QA and the sample efficiency (SE) of EAGER. The SE is defined as $SE = \frac{1}{n_{frames}} \sum_i^{n_{frames}} r_i$), where r_i is the extrinsic reward and n_{frames} the number of frames seen during training. Figure 15 shows once again that EAGER is robust to the quality of the QA module with an almost constant sample efficiency when $SR > 0.52$

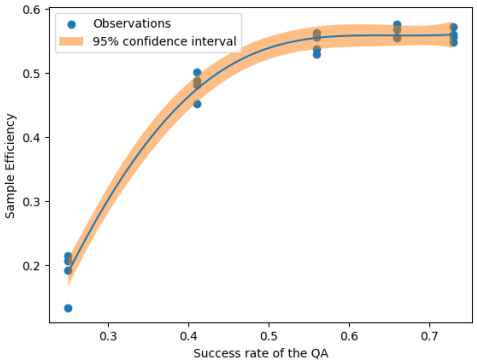

Figure 15: Gaussian process model to fit Sample Efficiency (SE) of the EAGER algorithm function of Success Rate (SR) of the QA. For each value of SR, the different points correspond to the different seeds used to train the RL agent on the PutNextTo task.

EAGER, a precise guiding technique To understand why EAGER achieves better results than ELLA or RIDE, we propose an answer based on qualitative observations. We believe that EAGER guides the agent more finely. The generated questions break down the trajectory into more precise key points than the auxiliary objectives generated in ELLA. To verify our point, we plot the average trajectory over 4 seeds for agents trained with EAGER or ELLA, see Figure 16. Looking at the average trajectories, it appears that EAGER generates trajectories that are spatially more compact and that return to the same place fewer times.

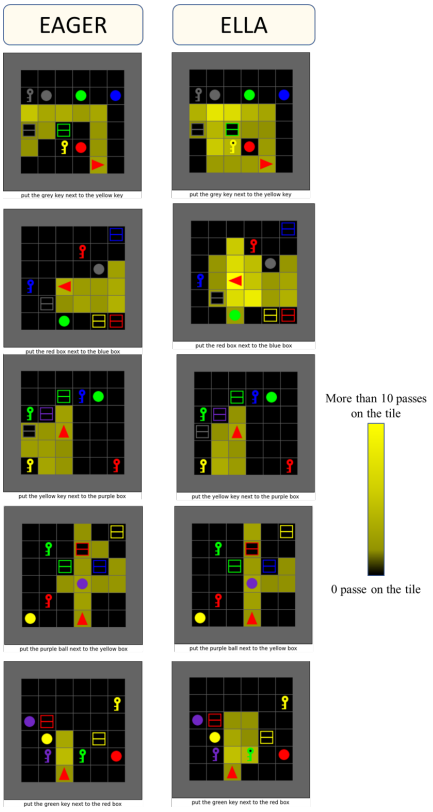

Figure 16: Average trajectory for PutNextTo-Local tasks for EAGER and ELLA. The colour of the tiles depends on the average number of passes of the agent on the tile. The 5 environments presented here have been randomly selected. The position of the agent is its position at the beginning of the episode.

D Commented EAGER algorithm

Algorithm 2 outlines the pseudo-code of our learning architecture. At the beginning of an episode, the EAGER algorithm uses the goal to generate an active set of questions $\mathcal{Q}$ that is updated at each time step by removing correctly answered questions. Each question can be understood as an auxiliary objective guiding the agent towards the completion of the main goal. When an auxiliary objective is completed (i.e. the corresponding question is correctly answered), the agent receives an intrinsic reward λ. To avoid modifying the optimal policy with these additional rewards, we substract the sum of discounted intrinsic rewards from the reward at the last step of successful trajectories with the NEUTRALISE function. In ELLA [26], the authors prove that shaping the reward does not change the optimal policy when using this neutralisation procedure.

Algorithm 2 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
    ω^g, o_0, done_0 ←ENV.reset()
    Q_0 = {q_1, ..., q_k} ← QG_k(ω^g)                ▷ Generate questions by masking words
    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)        ▷ Shape reward and update active set of question Q
        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)                                ▷ Answer q using the trajectory, τ = (o_i, a_i)_{i∈[|0,t|]}
        if ã* is correct answer to q then
            r'_t = r_t + λ QA(ã* | q, τ)
            Q = Q \{q}                               ▷ Update the active set of questions
        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             ▷ T_S time steps where the agent has received a shape reward
    return r'_N
end function
```

E Hyperparameters tables

This section contains three tables: the hyperparameters used for training the QA in Table 3, the hyperparameters for the PPO algorithm in Table 4, and the shape reward value λ depending on the task in Table 5.

Table 3: QA training hyperparameters

Variable	**Value**
batch size	10
learning rate (lr) at the beginning	10^{-4}
number of steps before decreasing lr	5
factor decrease	0.1

In Table 5 we give all the elements we use to compute λ, which is the value of the intrinsic reward: $\lambda = \frac{\gamma^N r_N}{k}$, where $\gamma = 0.99$ is the discount factor. $r_t = 20(1 - 0.9\,\frac{t}{H})$ is the reward obtained for completing the goal at step t, with H the maximum number of steps for a given task. In the calculation of λ, we assume that once trained, the agent completes the goal in N steps in the worst case scenario. k represents the maximum number of questions that can be generated from goals of a certain task.

Table 4: PPO hyperparameters

Variable	**Value**
batch size	2560
mini-batch size	1280
discount factor	0.99
lr	7×10^{-4}
entropy coefficient	0.01
loss coefficient	0.5
clipping-ϵ	0.2
generalised advantage estimation parameter	0.99

Table 5: Value of λ depending on the task.

Task	**k**	**H**	**N**	λ
PutNextTo-Local	4	128	40	2.4
PutNextTo-Medium	4	256	80	1.6
Unlock-Medium	2	128	40	4.8
Sequence-Medium	9	512	185	0.23