META-RL INDUCES EXPLORATION IN LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) has enabled the training of large language model (LLM) agents to interact with the environment and to solve multi-turn long-horizon tasks. However, the RL-trained agents often struggle in tasks that require active exploration and fail to efficiently adapt from trial-and-error experiences. In this paper, we present LAMER, a general Meta-RL framework that enables LLM agents to actively explore and learn from the environment feedback at test time. LAMER consists of two key components: *(i)* a cross-episode training framework to encourage exploration and long-term rewards optimization; and *(ii)* in-context policy adaptation via reflection, allowing the agent to adapt their policy from task feedback signal without gradient update. Experiments across diverse environments show that LAMER significantly improves performance over RL baselines, with 11%, 14%, and 19% performance gains on Sokoban, MineSweeper and Webshop, respectively. Moreover, LAMER also demonstrates better generalization to more challenging or previously unseen tasks compared to the RL-trained agents. Overall, our results demonstrate that meta-reinforcement learning provides a principled approach to induce exploration in language agents, enabling more robust adaptation to novel environments through learned exploration strategies.

1 INTRODUCTION

Recent advances in large language models (LLMs) have shifted from building conversational systems to decision-making agents capable of reasoning about and interacting with their environments (Yao et al., 2023b; Shinn et al., 2023; Wang et al., 2025; Feng et al., 2025). To accomplish the goal, language agents operate in multi-turn, textual observation-action loops, and must adapt quickly using the memory across turns. Central to such adaptation is *exploration*, which allows agents to test uncertain actions, acquire new knowledge, and avoid premature convergence on suboptimal strategies. However, unlike humans that can explore systematically and make fast adaptation in new environments (Wilson et al., 2014), LLM agents do not robustly engage in exploration without substantial interventions (Krishnamurthy et al., 2024).

Recent works has begun to address this limitation by guiding LLMs toward exploratory behaviors at test time. For example, Tajwar et al. (2025) train models offline to distill exploration strategies from trajectories from diverse environments, while Gandhi et al. (2024) induce such strategies from offline search traces. Setlur et al. (2025) train models to learn to explore in-context as a better way of spending test-time compute (Snell et al., 2025). However, these works either focus on single-turn non-agentic reasoning problems, or rely on offline data that limits them to imitation rather than active exploration.

In this work, we take a step toward agents that can *actively explore* their environment, gather feedback, and leverage this experience for more effective exploitation. Since multi-turn tasks often have a sparse success signal after an episode, we consider a multi-episode regime (Shinn et al., 2023) where an episode is the unit of exploration and exploitation. Balancing exploration and exploitation can then be naturally formulated as a *cross-episode* reinforcement learning (RL) framework. Training across many similar but different environments under this framework leads to *meta reinforcement learning* (Meta-RL) (Duan et al., 2016; Wang et al., 2016; Bauer et al., 2023; Beck et al., 2025), where the agent is forced to discover general strategies that work in unseen and potentially harder environments.

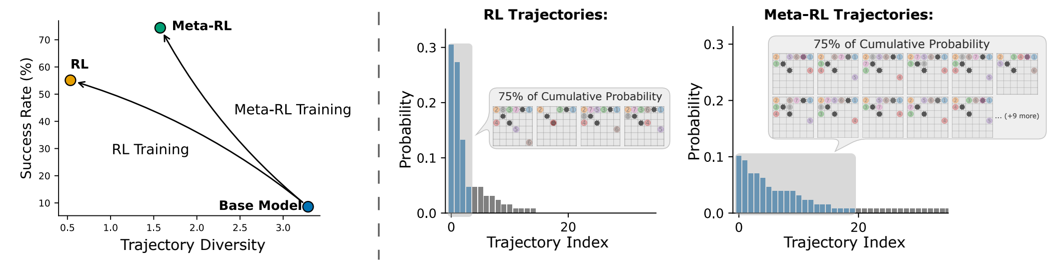


Figure 1: Comparison of RL and Meta-RL training on the Minesweeper environment. *Left*: Meta-RL training with LAMER retains higher sample diversity from the base model while achieving better success rates, reaching a better trade-off between exploration and exploitation. *Right*: Distinct trajectories and their empirical probabilities aggregated over multiple sampled trajectories in the MineSweeper environment. Each trajectory corresponds to a sequence of clicks (numbered cell) on the board. Sample diversity is quantified by the entropy of the empirical distribution. Meta-RL trained model produces more diverse and explorative trajectories.

Building upon this, we propose LAMER (**L**LM **A**gent with **Me**ta-**R**L), a general Meta-RL framework for LLM agent training. LAMER contains two important design factors. First, unlike standard single-episode RL, LAMER is designed around a multi-episode structure to train the agent to solve the problem through trial and error. In early episodes, the agent is encouraged to gather diverse experiences and useful information of the environment, which are used to adapt its policy in later episodes. By maximizing long-term rewards across episodes, the agent internalizes a learning algorithm that explicitly incentivizes exploration for improved downstream exploitation. Second, at both training and test time, the agent effectively leverages the feedback and reflection from the past episodes to determine the strategy for the next episode, which essentially implements an RL algorithm in context, and making the approach naturally suited for LLM agents. Meta-RL produces more diverse samples while simultaneously achieving higher performance, reaching a better balance between exploration and exploitation (Figure 1). To the best of our knowledge, this is the first time a meta-RL framework is used for LLM agent training.

We evaluate LAMER on four challenging long-horizon tasks: Sokoban (Racanière et al., 2017), MineSweeper (Li et al., 2024), Webshop (Yao et al., 2022) and ALFWorld (Shridhar et al., 2020). Using Qwen3-4B (Yang et al., 2025), we demonstrate that LAMER consistently outperforms prompting and RL baselines on all the environments. In addition, we observe that the trained model has learned a balance between exploration and exploitation, resulting in a better test-time scaling performance. In particular, LAMER adapts the trained policy at test time, with $11/14/19\%$ absolute performance gains on Sokoban, MineSweeper and Webshop, respectively. Furthermore, we show that LAMER trained model achieves a better generalization to harder and out-of-distribution tasks. In summary, LAMER takes a step toward autonomous agents that can actively act to uncover information and improve their decision-making in the new environments.

2 Related Work

LLM-as-agent. As LLMs become increasingly capable of reasoning about complex scenarios (Wei et al., 2022), there is a growing interest in making them decision-making autonomous agents. Earlier works rely on prompting frozen LLMs (Yao et al., 2023b; Shinn et al., 2023; Park et al., 2023; Wang et al., 2024a; AutoGPT). ReAct (Yao et al., 2023b) prompts LLMs with in-context examples to generate both textual actions and reasoning thoughts. Later, Reflexion (Shinn et al., 2023) extends this principle to the multi-episode setting, where the agent verbally reflects on the last episode and maintains their own reflection buffer for the next episodes. More recent research trains LLM agents through designing advanced RL algorithms (Wang et al., 2025; Feng et al., 2025) for multi-turn interactions, or supervised fine-tuning on generated interaction trajectories across diverse tasks (Tajwar et al., 2025). The evaluation of LLM agents also poses challenges because of fully verbal interactions with the environments. Recent benchmarks span a wide range of domains, including text-based embodied environments (Shridhar et al., 2020), e-commerce website (Yao et al., 2022), bandits (Nie et al., 2024), classic games (Park et al., 2025; Li et al., 2024) and other tasks (Liu et al., 2024;

Nathani et al., 2025). For a more comprehensive overview of these efforts, we refer readers to recent surveys (Wang et al., 2024b; Zhang et al., 2025).

Meta reinforcement learning (Meta-RL) (Beck et al., 2025) focuses on "learning to reinforcement learn" in order to rapidly adapt to new environments. Similar to meta-learning (Thrun & Pratt, 1998; Hospedales et al., 2021), it involves an inner-loop that represents an RL algorithm (i.e. adaptation strategy) by itself, together with an outer-loop that updates the meta-parameters so that the inner loop becomes more effective across many tasks. By training across many tasks, the outer loop forces the agent to learn the exploration strategy necessary to solve the tasks, while the inner loop enables the agent adapt quickly based on the exploration. Depending on how the inner loop is done, there are in-context methods and gradient-based methods. For example, Duan et al. (2016); Wang et al. (2016); Stadie et al. (2018) represent the inner-loop as a history-dependent policy parametrized by an RNN, the adaptation is thus done 'in-context' through gathering more information stored in the memory states. On the other hand, Finn et al. (2017) leverages a gradient-based approach in which the inner adapts a general meta-policy learned by the outer-loop. Our work lies in the former category, where the adaptation occurs entirely in-context at test time, naturally leveraging LLMs' in-context learning abilities.

Test-time compute. A Meta-RL framework in LaMer can be considered as amortizing the test-time compute by training tasks in a multi-episode manner rather than a single episode. In this way, the learned in-context policy adaptation balances exploration and exploitation for a fast adaptation at test time. This is essentially a better way of spending test-time compute (Snell et al., 2025; Muennighoff et al., 2025; Wu et al., 2025; Setlur et al., 2025). In our experiments, we match the training compute budget between an RL and a Meta-RL baseline, and show that meta-RL encourages superior test-time scaling behavior (through pass@k). Qu et al. (2025) similarly relates meta-RL to test-time compute, but they are limited to single-turn problems of mathematical reasoning, without leveraging the interactive feedback from the environment.

Reasoning in LLMs. More broadly, this work relates to reasoning in LLMs, because language agents must use reasoning as part of their decision-making. A large bulk of recent work on LLM reasoning has focused on more advanced prompting (Wei et al., 2022; Yao et al., 2023a), post-training (Cobbe et al., 2021; Luong et al., 2024; Shao et al., 2024; DeepSeek-AI et al., 2025) or bootstrapping (Zelikman et al., 2022) against verifiers or reward models, inducing structured search behavior (Gandhi et al., 2024; Moon et al., 2024), or reflecting on previous answers (Kumar et al., 2024; Xiong et al., 2025; Qu et al., 2024), etc. Most of these works focus on single-turn math (Hendrycks et al., 2021b; Cobbe et al., 2021) and coding (Chen et al., 2021; Hendrycks et al., 2021a) problems, while we target multi-turn agentic environments where environment feedback is available after every action and at the end of the episode.

3 Preliminaries

We consider the scenario where an LLM agent interacts with the environment to solve a multi-turn task. This process can be formulated as a Markovian decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma_{step})$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state space and action space, and R is the reward function. At each time step $t = 0, ..., T-1$, the LLM agent observes a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to its policy $a_t \sim \pi_\theta(\cdot|s_t)$. The environment then provides a scalar reward $r_t \in \mathbb{R}$ and transitions to the next state s_{t+1} according to the transition function $P(\cdot \mid s_t, a_t)$. A trajectory is the sequence of states, actions, and rewards over an episode, i.e., $\tau = (s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1})$. The objective of reinforcement learning is to maximize the expected discounted return:

$$\mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma_{step}^t r_t \right] \tag{1}$$

where $\gamma_{step} \in [0, 1]$ is the discount factor. Recent works (Wang et al., 2025; Feng et al., 2025) have shown that RL training has enabled LLM agents to interact with the environment and solve multi-turn tasks. However, such agents often learn a fixed policy during training and struggle to actively explore and adapt their behavior to the tasks at *test time* (Nie et al., 2024).

Meta-RL. Conversely, by training on a distribution of tasks, Meta-RL (Duan et al., 2016; Wang et al., 2016; Bauer et al., 2023; Beck et al., 2025) encourages exploration because it optimizes meta-parameters, such that the agent can solve new tasks quickly. In our case, the meta-parameters are the parameters of the LLM. This necessitates the agent to learn general exploration-exploitation strategies suitable for the task distribution trained on. For example, for most navigation tasks in partially observable environments, the optimal strategy is to gather the environment information and locate the target during the first episode, then reach the target as efficiently as possible in the second episode. This *explore-then-exploit* strategy implemented by the agent is itself a reinforcement learning algorithm, where the policy learned at the meta-level encodes how to adaptively switch between information-gathering and reward-maximizing behaviors depending on the stage of interaction with a new task. For LLM agents operating in multi-turn tasks, the policy can be in-context (*i.e.*, without parameter updates at test time), naturally leveraging the in-context capability of LLMs.

4 LaMer: A Meta-RL Framework for LLM Agents

Adopting the principle of Meta-RL, we present LaMer, a framework for training LLM agents with the ability to actively explore and adaptively learn from the environment. The framework addresses two central challenges. First, how to balance exploration and exploitation over multiple attempts at a task. To this end, LaMer introduces a cross-episode training scheme that treats each trial as a sequence of episodes, enabling the agent to explore in early episodes and exploit this information in later ones. Second, how to efficiently adapt the policy during training and evaluation. Instead of relying on gradient-based updates, LaMer uses self-reflection as an in-context adaptation mechanism, allowing the agent to summarize past experiences and adjust its strategy accordingly. Together, these two components enable scalable training of LLM agents under a unified Meta-RL framework, which can be optimized with standard RL algorithms.

Cross-episode training framework. In the training of LaMer, each trial consists of N episodes sequentially generated by the agent:

$$\mathcal{T} = (\tau^{(0)}, \tau^{(1)}, \dots, \tau^{(N-1)}), \quad \text{where } \tau^{(n)} \sim \pi_\theta^{(n)}(\cdot),\ n \in [0, N-1], \tag{2}$$

where $\pi_\theta^{(n)}(\cdot)$ is the policy at episode n updated from the accumulated history $\tau^{(0)}, ..., \tau^{(n-1)}$ through some adaptation strategy. For simplicity, in our analysis we assume all the episode contains the T steps of interactions with the environment, *i.e.*, $\tau^{(n)} = (s_0^{(n)}, a_0^{(n)}, r_0^{(n)}, ..., s_{T-1}^{(n)}, a_{T-1}^{(n)}, r_{T-1}^{(n)})$ for all $n \in [0, N-1]$. The rollout process terminates at n if $\tau^{(n)}$ is successful (as indicated by the environment feedback). Otherwise, the agent starts a new episode $\tau^{(n+1)}$ from the same initial state, repeating this procedure until the maximum episode budget is reached. For action $a_t^{(n)}$, the discounted return $g_t^{(n)}$ *within the episode* $\tau^{(n)} \in \mathcal{T}$ is:

$$g_t^{(n)} = \sum_{l=t}^{T-1} \gamma_{step}^{l-t} r_l^{(n)}, \tag{3}$$

where $\gamma_{step} \in [0, 1]$ is within-the-episode discount factor.

To enhance the exploration and maximize the long-term reward, in LaMer framework we define the discounted return $G_t^{(n)}$ *across the episodes* of $\mathcal{T}$ as:

$$G_t^{(n)} = \underbrace{g_t^{(n)}}_{\text{within-the-episode}} + \underbrace{\sum_{m=n+1}^{N-1} \gamma_{traj}^{m-n} g_0^{(m)}}_{\text{cross-episode}}, \tag{4}$$

where $\gamma_{traj} \in [0, 1]$ is the cross-episode discount factor. Finally, the LLM agent is trained via the following Meta-RL objective:

$$J(\theta) = \mathbb{E}_{\mathcal{T} \sim \pi_\theta} \left[\sum_{n=0}^{N-1} \gamma_{traj}^{n} \sum_{t=0}^{T-1} \gamma_{step}^{t} r_t^{(n)} \right] = \mathbb{E}_{\mathcal{T} \sim \pi_\theta} \left[G_0^{(0)} \right]. \tag{5}$$

Here, γ_{traj} is an important factor for the trade-off between *exploration* and *exploitation*. Ideally, small γ_{traj} biases the objective towards early episodes and will lead to rapid exploitation to solve the problem. In comparison, a larger γ_{traj} emphasizes long-horizon return and therefore encourages more exploration at the early stage.

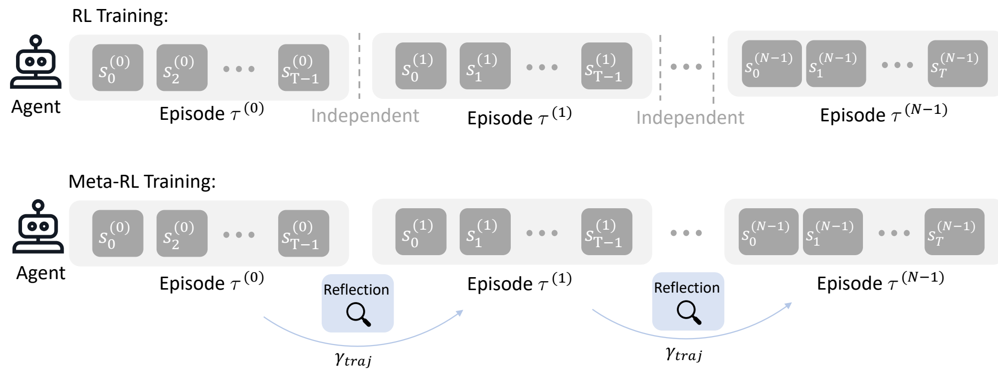


Figure 2: Comparison between the training processes of RL (top) and Meta-RL used in LaMer (bottom). For a single task, RL generates a group of trajectories independently. In contrast, in LaMer we use Meta-RL and produce the trajectories sequentially and adapt the policy in-context with self-reflection. Trajectory discount factor γ_{traj} is used for cross-episode credit assignment.

In-context policy adaptation with self-reflection. In Meta-RL, policy adaptation is the inner loop of the learning process of an LLM agent. Therefore, a flexible and efficient adaptation mechanism plays an important role during training and methods like gradient descent Finn et al. (2017) might be too expensive, especially for LLMs. In LaMer, we propose a self-reflection based strategy(Shinn et al., 2023) to *adapt the policy in-context* (Brown et al., 2020; Laskin et al., 2023). Specifically, after each episode finishes, we prompt the agent to generate the textual reflection on the previous attempt, providing specific feedback and plan to guide the next episode (see Appendix A for the used prompt). The policy is therefore updated through modifying the context, $\pi_\theta^{(n)}(\cdot) = \pi_\theta(\cdot|\mathcal{H}^{(n)})$ where $\mathcal{H}^{(n)}$ denotes the inter-episode memory that contains both the history trajectories and reflections. Importantly, the self-reflection step is also explicitly trained in LaMer using the reward obtained in the next episode. Note that the content $\mathcal{H}^{(n)}$ can be adjusted according to the predefined memory buffer to reduce the context length and improve the efficiency. By default, we retain both history and reflection in $\mathcal{H}^{(n)}$, and provide an ablation study in Section 6.2.

Comparison to RL training. Compared to the RL objective (Eq. 1), Meta-RL extends the credit assignment across multiple episodes to incentivize exploration in the early stages. In practice, given a single task, both RL and Meta-RL will sample a group of episodes during training to estimate the advantage. The key difference is that the RL rollouts are independent, whereas in Meta-RL each episode is conditioned on the preceding rollouts within the trial. Figure 2 illustrates the conceptual difference between the training processes of RL and Meta-RL.

Optimization. The proposed Meta-RL objective in (5) can be optimized with standard policy gradient methods. Given the per-action cross-episode return $G_t^{(n)}$ defined above, the gradient can be estimated by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{n=0}^{N-1}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^{(n)}|s_t^{(n)}, \mathcal{H}^{(n)})\, A_t^{(n)}\right], \tag{6}$$

where $A_t^{(n)}$ is the advantage estimation derived from $G_t^{(n)}$. The framework is compatible with widely used optimizers such PPO (Schulman et al., 2017) and critic-free approaches such as GRPO (Shao et al., 2024) and GiGPO (Feng et al., 2025).

5 Experiments

In this section, we conduct comprehensive experiments to evaluate LaMer Meta-RL framework. Specifically, we present the evaluation on: *(i)* the overall performance of LaMer across different agent environments; *(ii)* the generalization ability of LaMer to harder tasks; *(iii)* the generalization of LaMer under distribution shifts.

Table 1: Performance on Sokoban, MineSweeper and Webshop environments. The results of p@1, p@2 and p@3 denote the success rate (%) under 1, 2, and 3 attempts, respectively.

Method	Sokoban			MineSweeper			Webshop		
	p@1	p@2	p@3	p@1	p@2	p@3	p@1	p@2	p@3
Prompting									
Zero-shot	6.8	9.8	12.9	4.5	6.6	8.6	1.4	2.1	2.3
ReAct	7.2	9.6	12.5	6.3	7.0	10.9	3.1	4.5	4.5
Reflexion	6.4	9.8	12.1	5.5	7.2	9.8	2.7	3.3	3.5
Training with RL									
PPO	12.5	15.4	16.8	29.7	34.2	35.5	53.1	54.5	54.9
RLOO	13.5	16.6	18.8	48.8	51.2	51.6	67.6	68.4	69.1
GRPO	22.9	26.4	27.0	36.3	40.0	40.4	72.9	73.0	73.0
GiGPO	41.6	43.6	44.1	**52.0**	54.9	55.1	**73.4**	74.6	75.2
Training with Meta-RL (ours)									
LaMer	**42.4**	**52.0**	**55.9**	44.1	**66.4**	**74.4**	67.8	**84.4**	**89.1**

5.1 Experimental Setup

Environments. We evaluate LaMer on four challenging and diverse environments: Sokoban (Racanière et al., 2017), MineSweeper Li et al. (2024), Webshop Yao et al. (2022) and ALFWorld Shridhar et al. (2020). Among them, Sokoban is a classic grid-based game on planning where the environment is *fully observable*. In comparison, the environments of MineSweeper, ALFWorld and Webshop are *partially observable*, requiring the agent to explore and plan under uncertainty to finish the task. Specifically, MineSweeper is a board game about logical deduction on hidden cells. Webshop simulates realistic web-based shopping tasks, and ALFWorld provides text-based embodied environments. We provide the detailed explanation and prompts of the environment in Appendix A. All the experiments are conducted with the text modality, though the proposed method can be naturally applied to multimodal environments.

Training details. We use Qwen3-4B (Yang et al., 2025) as our base model for all the experiments. To improve rollout efficiency in agentic loops, we use the non-thinking mode during trajectory generation. Additionally, we validate our method on Llama3.1-8B-Instruct (Grattafiori et al., 2024), see Appendix D.1 for the results. For the Meta-RL setting, we use $\gamma_{traj} = 0.6$ as the default trajectory discount factor and explore its influence in the ablation study. We use GiGPO as the default optimization algorithm for all the environments with LaMer. Importantly, for Meta-RL training we sample $N = 3$ episodes and set group size to 8 for each task. To ensure a fair comparison, we use a group size of 24 in standard RL training, yielding the same number of trajectories used for each gradient update step. All other hyperparameters and configurations are kept identical across RL and Meta-RL for a fair comparison and are provided in Appendix C.

5.2 Performance comparison

In this section, we compare the performance of our proposed algorithm LaMer with prompting-based baselines (Zero-shot, ReAct (Yao et al., 2023b; Shinn et al., 2023)), and RL methods (PPO (Schulman et al., 2017), RLOO (Ahmadian et al., 2024), GRPO (Shao et al., 2024), and GiGPO (Feng et al., 2025)) across three environments: Sokoban, MineSweeper, and Webshop. For each method, we report the success rates under 1, 2, and 3 attempts (*i.e.*, pass@1, pass@2, and pass@3, respectively). The results are summarized in Table 1.

Meta-RL obtains better performance. Across all three environments, LaMer trained with Meta-RL consistently outperforms both prompting-based baselines and RL-training methods on the final pass@3 success rate. On Sokoban, LaMer achieves a 55.9% pass@3 success rate, substantially outperforming the 44.1% from the strongest RL baseline (GiGPO) and 12.9% from prompting-based methods. Similarly, on MineSweeper LaMer reaches 74.4% pass@3 success rate, which is 19% higher than the best RL-trained model. On Webshop, LaMer also performs 14% better than the RL-trained methods. Notably, the performance gains are not limited to pass@3: improvements

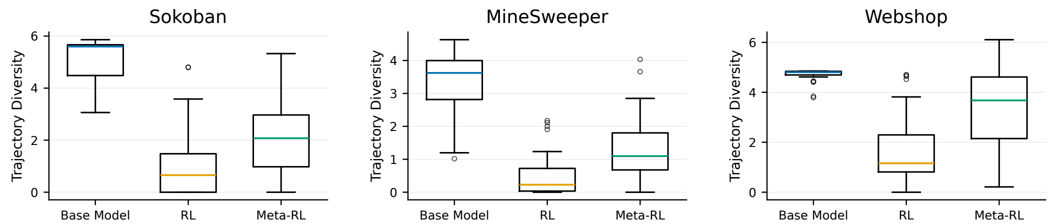


Figure 3: Trajectory diversity of base and trained models. Compared to RL, Meta-RL preserves more diverse trajectories from the base model, striking a better balance between exploration and exploitation.

are also observed on pass@2 for all the environments, and even pass@1 for Sokoban. Together, these results demonstrate that LaMer delivers consistent benefit on the trained agents to solve the long-horizon task in the complex environments.

Meta-RL exhibits stronger test-time scaling. Beside achieving the best final pass@3 performance, Meta-RL also demonstrates remarkable effectiveness in test-time scaling, with larger performance gains across attempts according to the results in Tabel 1. For example, the improvement from pass@1 to pass@3 on Sokoban is 13.5% in Meta-RL, significantly larger both RL-trained and prompting-based baselines (which are less than 5%). Notably, although Meta-RL starts with slightly lower pass@1 performance than RL baseline (GiGPO) in MineSweeper and Webshop, it quickly recovers and surpasses all baselines by pass@2 and pass@3. The results indicate that the trained model has successfully learned to actively explore in the earlier episodes and adapt effectively from the mistakes, leading to significant gains in the subsequent attempts. The illustrative trajectories and reflections produced by the trained agents are presented in Figure 6 in Appendix E.

Meta-RL induces exploration. To further analyze the behavior of the models, we measure the diversity of answer trajectories across environments. For each question, we sample multiple trajectories from the agent and group the identical trajectories that have the same states and actions. These groups are used to form the empirical distribution over distinct trajectories, as shown in Figure 1. We then estimate the entropy the distribution to quantify the *trajectory diversity*. Figure 3 compares the trajectory diversity of the base model, RL, and Meta-RL agents across environments. We observe that the base model exhibits the highest entropy, indicating it generates a wide range of trajectories, though this diversity does not translate into higher success rates (see Table 1). RL-trained agent reduces diversity and converges toward more deterministic behaviors. In contrast, LaMer preserves a higher level of diversity than RL baselines, allowing more exploration at test time.

5.3 Generalization to Harder Tasks

Next we study the generalization ability of the pretrained models on harder tasks. To this end, we take the models trained with RL and Meta-RL and evaluate them on the harder tasks in the environments of Sokoban and MineSweeper. We increase the difficulty by using more boxes for Sokoban and more mines for MineSweeper in the grid. The results are shown in Figure 4. As

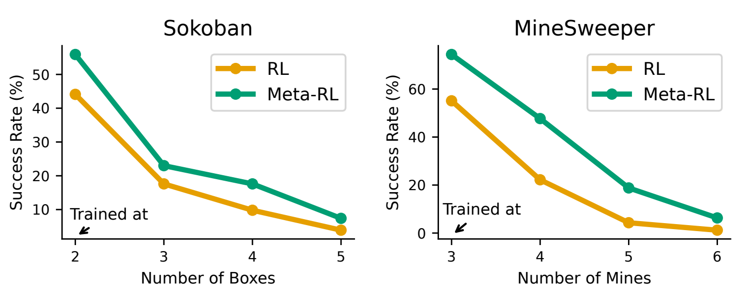


Figure 4: Performance of RL and Meta-RL trained model on the tasks with increased difficulty. For Sokoban, we gradually increase the number of boxes and for Minesweeper, we increase the number of mines in the grid.

expected, the model trained with both RL and Meta-RL underperforms on harder tasks with an increasing number of boxes or mines in the grid. However, Meta-RL consistently outperforms RL on all the difficulty levels. Notably, on the most difficult setting, the model trained from Meta-RL still outperforms the RL-trained model with 10% performance gap on Sokoban, and 5% performance gap on the MineSweeper. The consistent gap indicates that LAMER trained with Meta-RL not only performs better on the training distribution, but also generalizes better to the harder tasks.

5.4 GENERALIZATION TO UNSEEN TASKS

We further study the ability LAMER and alternative methods to generalize out-of-distribution. For this experiment, we use the ALFWorld environment (Shridhar et al., 2020). As a text-based embodied environment, ALFWorld contains 6 categories of common household activities: Pick and Place (*Pick*), Examine in Light (*Look*), Clean and Place (*Clean*), Heat and Place (*Heat*), Cool and Place (*Cool*), and Pick Two and Place (*Pick2*). We use the tasks of *Pick*, *Look*, *Clean* and *Heat* as in-distribution tasks and use *Cool* and *Pick2* as out-of-distribution tasks. We train LAMER and alternative baselines with instances from in-distribution tasks and then evaluate the model on both in-distribution tasks (with held-out test set), and the examples of out-of-distribution tasks. The results are shown in Table 2. As we can see, RL trained model generally performs well on in-distribution tasks and outperforms prompting-based methods by achieving more than 20% improvement on *Look*, *Clean* and *Heat*. However, on out-of-distribution tasks, *Cool* and *Pick2*, it only obtains 58.1% and 36.0% success rate. Meta-RL consistently outperforms RL on both in-distribution and out-of-distribution tasks, with a notable performance gap on out-of-distribution tasks. In particular, our LAMER framework achieves 23% performance gains on Cool and around 14% on *Pick2*. Overall, these results suggest that on ALFWorld, Meta-RL trained model could generalize better to out-of-distribution tasks compared to the RL trained model.

Table 2: Evaluation of out-of-distribution generalization on the tasks of ALFWorld.

Method	*i.d*				*o.o.d*	
	Pick	Look	Clean	Heat	Cool	Pick2
Prompting	91.9	52.9	48.4	44.8	42.8	21.2
RL	95.5	83.0	67.9	86.6	58.1	36.0
Meta-RL	**97.7**	**100.0**	**90.2**	**89.5**	**81.0**	**50.2**

6 ANALYSIS

We further conduct a series of ablation studies on the key design factors of LAMER, including *(i)* the influence of trajectory discounted factor γ_{traj} on the trade-off between exploration and exploitation and *(ii)* the ablation of inter-episode memory configurations. We additionally discuss *(iii)* the computation budget of the proposed Meta-RL framework compared to the RL training.

6.1 INFLUENCE OF TRAJECTORY DISCOUNT FACTOR

The cross-episodes discount factor γ_{traj} controls how rewards are propagated within a trial, thereby mediating the balance between exploration and exploitation in the LAMER framework during training. To understand the effect of the discount factor, we train the agents with LAMER using different values of γ_{traj} on Sokoban, MineSweeper and Webshop. The results are shown in Figure 5. We observe that a larger value of γ_{traj} does not necessarily lead to better final performance on pass@3, instead, the optimal setting of γ_{traj} varies across different environments. For Sokoban and Webshop, intermediate values like $\gamma_{traj} = 0.6$ yield the best results, suggesting that balancing immediate and long-term rewards is more important for these tasks. In contrast, MineSweeper benefits from relatively larger γ_{traj} like 0.9, indicating that extended credit assignment better supports strategic exploration in this environment. Overall, the results show that γ_{traj} provides a practical way to control the trade-off between exploration and exploitation across environments.

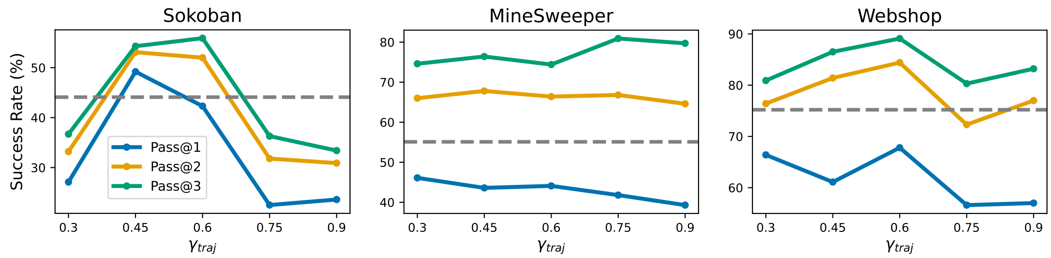


Figure 5: Success rates of models trained with different γ_{traj}. A higher value encourages more exploration during training.

6.2 Ablation on the inter-episode memory

In LAMER, the agent policy is adapted *in-context* through the inter-episode memory $\mathcal{H}^{(n)}$, which by default contains both the trajectories and reflections of previous episodes. To assess the influence of memory content to the training, we consider two alternative configurations in $\mathcal{H}^{(n)}$: (1) only history trajectories; (2) only reflections. The performance of the trained agents in each configurations are reported in Table 3. The results show that self-reflection provides a clear benefit in LAMER, leading to 21.6% improvement on Sokoban, 11.0% on Minesweeper and 3.5% on Webshop, respectively. Interestingly, the reflection-only configuration also outperforms the default setting in LAMER (which contains both trajectory and reflection) across all environments. We hypothesize that this is because reflection-only memory presents more concise and focused guidance, leading to more effective adaptation of the agent's behavior.

Table 3: Comparison of LAMER with different inter-episode memory configurations.

Content in $\mathcal{H}^{(n)}$	Sokoban	MineSweeper	Webshop
Trajectory-only	34.8	69.5	89.3
Reflection-only	**56.4**	**80.5**	**92.8**
Both	55.9	74.4	89.1

6.3 Training budget

We conclude with a discussion on the training budget of RL and Meta-RL, focusing on both data usage and computational efficiency. To ensure a fair comparison, we set the group size for standard RL to be three times larger than that of Meta-RL. This adjustment guarantees that the two methods consume the same number of trajectories for each gradient update. Aside from this scaling, all other experimental configurations—such as learning rates, batch sizes, and network architectures—are held constant. This design choice highlights that Meta-RL does not require a larger data budget compared to RL; in other words, both methods rely on the same total number of trajectories to learn.

Nevertheless, LAMER might still introduce additional training time cost compared to the RL baselines. In RL training, all the episodes could be sampled in parallel since they are independent. In contrast, LAMER exhibits less parallelism since episodes within the same trial needs to be generated sequentially. As a result, we observe around twice the training time cost for LAMER in our current implementation. This suggests that more sophisticated sampling strategies, such as asynchronous rollout, could further improve the efficiency of LAMER for training LLM agents.

7 Conclusion

Being able to explore and gather information from the environment is crucial in building autonomous agents that can adapt quickly and robustly. We introduced LAMER, a general LLM agent training framework leveraging the principle of meta reinforcement learning. Unlike previous RL methods that maximize a single-episode return for immediate payoff, LAMER maximizes a discounted cross-episode return, naturally balancing when to explore versus when to exploit to maximize long-term performance. The exploration allowed at training time teaches the agent general explorative strategies that enable a rapid in-context adaptation at test time. We show across diverse environments that LaMer significantly outperforms RL methods, is able to generalize to harder environments, and scales better with more episodes at test time.

Limitations and future work. Our results raise several promising directions for future work. (1) The generality of our method allows for combining it with other RL algorithms or self-reflection frameworks. We hypothesize that a more advanced advantage estimation strategy or a stronger reasoning model may enhance the performance. (2) Our approach requires sampling episodes sequentially for rollouts since episodes are dependent in cross-episode training. This eventually leads to longer training time than RL methods. More efficient training strategies will be explored in future work. (3) Finally, LaMer trained on easier environments can generalize to harder environments of the same kind or relatively similar domains. This ultimately suggests possibilities in building generalist agents that can adapt to completely novel environments.

LLM Usage Statement

LLM is mainly used for proofreading and as a plot assistant in this work.

Reproducibility Statement

In order to ensure that our work is reproducible, we have provided experimental details in Section 5.1, together with the template of prompts we used in Appendix B. Complete code documentation is under development and will be made available alongside the paper's final version.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

AutoGPT. Significant-gravitas/auto-gpt: An experimental open-source attempt to make gpt-4 fully autonomous., 2023. URL `https://github.com/Significant-Gravitas/Auto-GPT/tree/master`.

Jakob Bauer, Kate Baumli, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, et al. Human-timescale adaptation in an open-ended task space. In *International Conference on Machine Learning*, pp. 1887–1935. PMLR, 2023.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A tutorial on meta-reinforcement learning. *Foundations and Trends® in Machine Learning*, 18(2–3):224–384, 2025. ISSN 1935-8245.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rlˆ2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Kanishk Gandhi, Denise H J Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah Goodman. Stream of search (sos): Learning to search in language. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=2cop2jmQVL`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan

Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps, 2021a. URL `https://arxiv.org/abs/2105.09938`.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b. URL `https://openreview.net/forum?id=7Bywt2mQsCe`.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9): 5149–5169, 2021.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? *Advances in Neural Information Processing Systems*, 37:120124–120158, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Stenberg Hansen, Angelos Filos, Ethan Brooks, maxime gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=hy0a5MMPUv`.

Yinghao Li, Haorui Wang, and Chao Zhang. Assessing logical puzzle solving in large language models: Insights from a minesweeper case study. *CoRR*, abs/2311.07387, 2023. doi: 10.48550/ARXIV.2311.07387. URL `https://doi.org/10.48550/arXiv.2311.07387`.

Yinghao Li, Haorui Wang, and Chao Zhang. Assessing logical puzzle solving in large language models: Insights from a minesweeper case study. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 59–81, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.4. URL `https://aclanthology.org/2024.naacl-long.4/`.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning with reinforced fine-tuning, 2024. URL https://arxiv.org/abs/2401.08967.

Seungyong Moon, Bumsoo Park, and Hyun Oh Song. Guided stream of search: Learning to better search with language models via optimal path guidance, 2024. URL https://arxiv.org/abs/2410.02992.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. Mlgym: A new framework and benchmark for advancing ai research agents, 2025. URL https://arxiv.org/abs/2502.14499.

Allen Nie, Yi Su, Bo Chang, Jonathan N Lee, Ed H Chi, Quoc V Le, and Minmin Chen. Evolve: Evaluating and optimizing llms for exploration. *arXiv preprint arXiv:2410.06238*, 2024.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, Ameya S. Mahabaleshwarkar, Bilal Kartal, Pritam Biswas, Yoshi Suhara, Kangwook Lee, and Jaewoong Cho. Orak: A foundational benchmark for training and evaluating llm agents on diverse video games. 2025. arXiv:2506.03610.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=DRC9pZwBwR.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement finetuning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=TqODUDsU4u.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amrith Setlur, Matthew YR Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms. *arXiv preprint arXiv:2506.09026*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

Fahim Tajwar, Yiding Jiang, Abitha Thankaraj, Sumaita Sadia Rahman, J Zico Kolter, Jeff Schneider, and Russ Salakhutdinov. Training a generally curious agent. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=UeB3Hdrhda.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024b.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Robert C Wilson, Andra Geana, John M White, Elliot A Ludvig, and Jonathan D Cohen. Humans use directed and random exploration to solve the explore–exploit dilemma. *Journal of experimental psychology: General*, 143(6):2074, 2014.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2025. URL https://arxiv.org/abs/2408.00724.

Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. Self-rewarding correction for mathematical reasoning, 2025. URL https://arxiv.org/abs/2502.19613.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

APPENDIX

A TASK DESCRIPTION AND DETAILS

Sokoban. We include the classic video game Sokoban as a fully-observable environment. The game is a 2D square board, with N boxes scattered on the board. There are also N target positions marked on the board. The player is placed at an initial position, and the goal is to push all the boxes to the target positions. There is no correspondence between each box and the target position. When the player walks into a box, it gets pushed in that direction (if there's space). Boxes can't be pushed into walls or other boxes. Once a box is pushed into a corner or against a wall with no way to get behind it, it might become permanently stuck. There is no pull operation in this game. The agent, therefore, has to think several moves ahead to avoid getting boxes stuck in positions where they can't reach their targets. The difficulty of this task is controlled by the board size, the number of boxes, and the wall structure of the board. We train on a board size 6×6 with 2 boxes.

Minesweeper. We include the classic video game Minesweeper as a partially-observable environment. The game is a 2D square board, with several mines randomly scattered in the board cells. The goal of the game is to open all the safe cells without revealing the hidden mines. In each step, the agent opens a cell, and the first step is always safe. If a mine is revealed, the task ends in failure immediately. The state of the opened (safe) cells can either be empty or a number from 1 to 8, and the number specifies how many mines are adjacent to the specific cell. The agent needs to use the numbers marked on the opened cells to reason about the position of mines. Success is achieved when all safe cells are revealed. Our implementation is based on a simplified version of Li et al. (2023). The difficulty of this task is controlled by the board size and the number of mines. We train on a board size 6×6 with 3 hidden mines.

WebShop (Yao et al., 2022). We include WebShop as a partially-observable text-based environment that simulates online shopping. The agent is given a natural language instruction specifying a product to purchase with certain attributes. The environment presents a simplified e-commerce interface where the agent can search for products, navigate through search results, and examine product pages with details like price, color, size, and customer reviews. The agent must interpret the instruction, search effectively, filter through multiple product options, and select the item that best matches the specified criteria. Success is measured by whether the final purchased item satisfies all the requirements in the original instruction.

ALFWorld (Shridhar et al., 2020). We include ALFWorld as a partially-observable text-based environment that simulates household tasks in interactive fiction format. The agent receives natural language instructions for common household activities. The environment provides text descriptions of rooms, objects, and possible actions, while the agent must navigate through a house, interact with objects, and complete multi-step tasks. Objects may need to be found, picked up, cleaned, heated, or combined with other objects to achieve the goal. The agent's view is limited to the current room and nearby objects, requiring exploration and memory of previously visited locations. Success requires understanding the instruction, planning a sequence of actions, and executing them correctly while managing partial observability. We train on the training examples of the activities 'Pick', 'Look', 'Clean', 'Heat'. We evaluate in-distribution on the test examples from the same activities, and we evaluate out-of-distribution on the test examples from 'Cool', 'Pick2' activities.

B Example Prompts

We provide examples of prompts for each task. There are two types of prompts: (1) the standard version (with name 'Standard Prompt') used for prompting the agent to play the game; (2) the reflection prompt used for self-reflection on a past experience (with name 'Reflection Prompt')

There are variables such as {`past_experience_reflection`}, {`history_actions`} in the prompts, among with other task-specific hyperparameters. They are omitted in the prompts for clarity. In practice, they will be replaced with the actual content. Note that the {past_experience_reflection} will be empty for the first episode.

Similar to (Feng et al., 2025), we use `<action> </action>` block to indicate the final decision of the action, and we use `<remark> </remark>` to indicate the content of the self-reflection.

(see next page for the prompts)

B.1 Sokoban

Sokoban Standard Prompt

You are an expert agent operating in the Sokoban environment.

Symbols and Their Meaning
- Walls (#): These block movement. You can't move through or push anything into walls.
- Floor (_): Open spaces where you can walk and move boxes.
- Targets (O): The spots where boxes need to go.
- Boxes (X): These are what you need to push onto the targets.
- Player (P): That's you! You'll move around the grid to push boxes.
- Box on Target (✓): A box successfully placed on a target.
- Player on Target (S): You standing on a target.

Goal
Your goal is to push all the boxes (X) onto the target spots (O). Once all boxes are on the targets, you win!

Rules
Your admissible actions are ["up", "down", "left", "right"].
You can only push one box at a time. You can't pull boxes, so plan ahead to avoid getting stuck.
You can't walk through or push boxes into walls (#) or other boxes.
To avoid traps, do not push boxes into corners or against walls where they can't be moved again.
{example}

Observations
The initial state of the game is:

```
0: # # # # # #
1: # # # _ O #
2: # _ O _ _ #
3: # _ _ X X #
4: # _ # P _ #
5: # # # # # #
```

{past_experience_reflection}
You have already taken the following actions:
{history_actions}
Your current observation is:

```
0: # # # # # #
1: # # # _ O #
2: # _ O _ X #
3: # _ X P _ #
4: # _ # _ _ #
5: # # # # # #
```

Now it's your turn to make moves (choose the next {num_actions_per_turn} actions).
- Your response first be step-by-step reasoning about the current situation — observe the positions of boxes and targets, plan a path to push a box toward a target, and avoid traps like corners or walls.
- Then choose {num_actions_per_turn} admissible actions and present them within <action> </action> tags (separated by comma).

Sokoban Reflection Prompt

You are an expert agent operating in the Sokoban environment.

Symbols and Their Meaning
- Walls (#): These block movement. You can't move through or push anything into walls.
- Floor (_): Open spaces where you can walk and move boxes.
- Targets (O): The spots where boxes need to go.
- Boxes (X): These are what you need to push onto the targets.
- Player (P): That's you! You'll move around the grid to push boxes.
- Box on Target (✓): A box successfully placed on a target.
- Player on Target (S): You standing on a target.

Your Goal
Your goal is to push all the boxes (X) onto the target spots (O). Once all boxes are on the targets, you win!

Rules
Your admissible actions are ["up", "down", "left", "right"].
You can only push one box at a time. You can't pull boxes, so plan ahead to avoid getting stuck.
You can't walk through or push boxes into walls (#) or other boxes.
To avoid traps, do not push boxes into corners or against walls where they can't be moved again.

Your Task
You will be given the history of a past experience.
Your job is to **reflect on the past sequence**, identify any **mistakes or inefficiencies**, and then devise a **concise, improved plan** starting from the original initial state.

Past Experience
The initial state of the game is:

```
0: # # # # # #
1: # # # _ O #
2: # _ O _ _ #
3: # _ _ X X #
4: # _ # P _ #
5: # # # # # #
```

You have taken the following actions:
{history_actions}
The final state is:

```
0: # # # # # #
1: # # # _ O #
2: # _ O _ X #
3: # _ X P _ #
4: # _ # _ _ #
5: # # # # # #
```

The task is NOT successfully completed.
Now it's your turn to reflect on the past experience and come up with a new plan of action.
- Your response should first be step-by-step reasoning about the strategy and path you took to attempt to complete the task. Identify where things went wrong or could be better.
- Then devise a concise, new plan of action that accounts for your mistake with reference to specific actions that you should have taken.
- Finally, end the response with your reflection and improved plan inside <remark> </remark> tags, to guide the next trial.

B.2 MINESWEEPER

Minesweeper Standard Prompt

You are an expert agent operating in the Minesweeper game.
You will be given a two dimensional {board_size} by {board_size} board, with {n_mines} hidden mines.
The rows and columns are indexed from 1 to {board_size}.

Cell States
- Unopened cells (?): cells that are yet to be revealed and may contain a mine.
- Blank cells (.): opened and non-mine cells, and they have no neighboring mines
- Numbered cells (1-8): opened and non-mine cells, and the number indicates how many mines are in the eight neighboring cells, including those diagonally adjacent. For example, a cell with a '8' means all its neighboring cells contain mines.
- Mine cells (*): opened cells that contain a mine.

Your Goal
Your goal is to clear the board by revealing all the cells that don't contain mines, without detonating any of the hidden mines scattered throughout the board.
Use clues about the number of neighboring mines in each field to reason about the position of mines and non-mine cells.

Reveal Rules
Your admissible action is to choose ONE unopened cell (?) to reveal per turn. The outcome depends on the content of that cell:
- Blank cell (.): That cell is revealed, and all contiguous blank cells plus their bordering numbered cells are automatically revealed (auto-cascade).
- Numbered cell (1–8): Only that single cell is revealed, showing the count of neighboring mines.
- Mine (*): The game ends immediately in a loss.
Observation
The initial state of the game is:

```
Row 1: . . . . . .
Row 2: . . . 1 1 1
Row 3: . . . 1 ? ?
Row 4: 1 1 . 1 2 ?
Row 5: ? 1 . . 1 1
Row 6: ? 1 . . . .
```

{past_experience_reflection}
You have already chosen the following cells to reveal: (6, 1)
Your current observation is:

```
Row 1: . . . . . .
Row 2: . . . 1 1 1
Row 3: . . . 1 ? ?
Row 4: 1 1 . 1 2 ?
Row 5: ? 1 . . 1 1
Row 6: 1 1 . . . .
```

Now it's your turn to make a move.
- Your should first reason step-by-step about the current situation — observe the status of the board, inferring the states of unopened cells (?).
- Then choose ONE unopened cell (?) to reveal. Put the index of cell in the format of "(row, col)" within the `<action> </action>` tag.

Minesweeper Reflection Prompt

You are an expert agent operating in the Minesweeper game.
You will be given a two dimensional {board_size} by {board_size} board, with {n_mines} hidden mines.
The rows and columns are indexed from 1 to {board_size}

Cell States
- Unopened cells (?): cells that are yet to be revealed and may contain a mine.
- Blank cells (.): opened and non-mine cells, and they have no neighboring mines
- Numbered cells (1-8): opened and non-mine cells, and the number indicates how many mines are in the eight neighboring cells, including those diagonally adjacent. For example, a cell with a '8' means all its neighboring cells contain mines.
- Mine cells (*): opened cells that contain a mine.

Your Goal
Your goal is to clear the board by revealing all the cells that don't contain mines, without detonating any of the hidden mines scattered throughout the board.
Use clues about the number of neighboring mines in each field to reason about the position of mines and non-mine cells.

Reveal Rules
Your admissible action is to choose ONE unopened cell (?) to reveal per turn. The outcome depends on the content of that cell:
- Blank cell (.): That cell is revealed, and all contiguous blank cells plus their bordering numbered cells are automatically revealed (auto-cascade).
- Numbered cell (1–8): Only that single cell is revealed, showing the count of neighboring mines.
- Mine (*): The game ends immediately in a loss.

Your Task
You will be given the history of a past experience.
Your job now is to **reflect on the past experience**, identify any **mistakes or inefficiencies**, and then devise a **concise, improved plan** for your next try starting from the original initial state.
Past Experience
The initial state of the game is:

```
Row 1: . . . . . .
Row 2: . . . 1 1 1
Row 3: . . . 1 ? ?
Row 4: 1 1 . 1 2 ?
Row 5: ? 1 . . 1 1
Row 6: ? 1 . . . .
```

You have chosen the following cells to reveal:
{history_actions}
The final state is:

```
Row 1: . . . . . .
Row 2: . . . 1 1 1
Row 3: . . . 1 ? ?
Row 4: 1 1 . 1 2 ?
Row 5: * 1 . . 1 1
Row 6: 1 1 . . . .
```

The task is NOT successfully completed.
Now it's your turn to reflect on the past experience and come up with a new plan of action.
- Your response should first be step-by-step reasoning about the strategy and path you took to attempt to complete the task. Identify where things went wrong or could be better.
- Then devise a concise, new plan of action that accounts for your mistake with reference to specific actions that you should have taken.
- Finally, end the response with your reflection and improved plan inside `<remark> </remark>` tags, to guide the next trial.

B.3 WEBSHOP

WebShop Standard Prompt

You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to: Find me slip resistant, non slip men's loafers & slip-ons with rubber outsole, rubber sole with color: 1877blue, and size: 11.5, and price lower than 70.00 dollars.
{past_experience_reflection}{history_actions}
Your admissible actions of the current situation are:
'search[your query]',
'click[search]'.
Now it's your turn to take one action for the current step.
Your response should first be step-by-step reasoning about the current situation, then think carefully which admissible action best advances the shopping goal.
Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

WebShop Reflection Prompt

You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to: Find me slip resistant, non slip men's loafers & slip-ons with rubber outsole, rubber sole with color: 1877blue, and size: 11.5, and price lower than 70.00 dollars.
You will be given the history of a past experience.
Your job is to **reflect on the past sequence**, identify any **mistakes or inefficiencies**, and then devise a **concise, improved plan** starting from the original initial state.
Below are the last few actions and corresponding observations you have:
{history_actions}
The task is NOT successfully completed.
Now it's your turn to reflect on the past experience and come up with a new plan of action.
- Your response should first be step-by-step reasoning about the strategy and path you took to attempt to complete the task. Identify where things went wrong or could be better.
- Then devise a concise, new plan of action that accounts for your mistake with reference to specific actions that you should have taken.
- Finally, end the response with your reflection and improved plan inside `<remark> </remark>` tags, to guide the next trial.

B.4 ALFWORLD

ALFWorld Standard Prompt

You are an expert agent operating in the ALFRED Embodied Environment.
-= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.

Your task is to: put a mug in desk.

{past_experience_reflection}{history_actions}
Your admissible actions of the current situation are:
'go to bed 1',
'go to desk 1',
'go to desk 2',
'go to drawer 1',
'go to drawer 2',
'go to drawer 3',
'go to drawer 4',
'go to drawer 5',
'go to drawer 6',
'go to garbagecan 1',
'go to laundryhamper 1',
'go to safe 1',
'go to shelf 1',
'go to shelf 2',
'go to shelf 3',
'go to shelf 4',
'go to shelf 5',
'go to shelf 6',
'inventory',
'look'.
Now it's your turn to take an action.
- Your response should first by step-by-step reasoning about the current situation.
- Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

ALFWorld Reflection Prompt

You are an expert agent operating in the ALFRED Embodied Environment.
-= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.

Your task is to: put a mug in desk.

You will be given the history of a past experience.
Your job is to **reflect on the past sequence**, identify any **mistakes or inefficiencies**, and then devise a **concise, improved plan** starting from the original initial state.

Below are the actions you took and the corresponding observations:
{history_actions}
The task is NOT successfully completed.
Now it's your turn to reflect on the past experience and come up with a new plan of action.
- Your response should first be step-by-step reasoning about the strategy and path you took to attempt to complete the task. Identify where things went wrong or could be better.
- Then devise a concise, new plan of action that accounts for your mistake with reference to specific actions that you should have taken.
- Finally, end the response with your reflection and improved plan inside `<remark>` `</remark>` tags, to guide the next trial.

C TRAINING DETAILS

LAMER is compatible with standard policy gradient algorithms. Without specification, we use GiGPO as the default optimization algorithm. During training, the self-reflection step is also explicitly trained using the reward in the subsequent episodes. During training, we match the total number of experiences sampled for each example between RL and Meta-RL to ensure a fair comparison. Specifically, for each sample $N = 3$ episodes and set group size to 8 for Meta-RL, and use a group size of 24 for standard RL training. Besides that, other hyper-parameters and configuration are kept the same between RL and Meta-RL training. We use Qwen3-4B as the base model and train it with Adam optimizer and a learning rate of $1e-6$. For Sokoban and MineSweeper, we train the agents with a batch size of 16 for 300 epochs. In comparison, we use batch size of 8 and 150 epochs for Webshop and ALFWorld. The environment reward is set to be 10 for successful trajectories and 0 for unsuccessful ones. We use temperature of 1.0 during rollout and 0.7 during evaluation. The maximum number of output tokens is set to 1024. Our code is based on the training framework of verl (Sheng et al., 2025) and verl-agent Feng et al. (2025).

D ADDITIONAL RESULTS

D.1 EXPERIMENTS ON DIFFERENT ARCHITECTURE

LAMER is designed as a general framework built on meta-RL principles and is model-agnostic. To validate this, we additionally ran experiments on Llama3.1-8B-Instruct (Grattafiori et al., 2024), showing that our method works well on models in a different model architecture and model size.

Table 4: Performance on Sokoban and MineSweeper environments using Llama3.1-8B-Instruct as base model. The results of p@1, p@2 and p@3 denote the success rate (%) under 1, 2, and 3 attempts, respectively.

Method	Sokoban			MineSweeper		
	p@1	p@2	p@3	p@1	p@2	p@3
GiGPO	5.9	6.3	6.3	39.1	41.4	42.2
LAMER	**17.6**	**19.9**	**20.3**	**47.7**	**61.3**	**65.6**

In Table 4, we compare LAMER with the strongest baseline GiGPO (Feng et al., 2025), on Sokoban and Minesweeper. We show The results indicate that on Llama-3.1-8B-Instruct, LaMer still outperforms the RL baselines across the environments, demonstrating the general applicability of our method to different model architecture and size.

D.2 COMPARISON TO RL BASELINES WITH INTER-EPISODE MEMORY

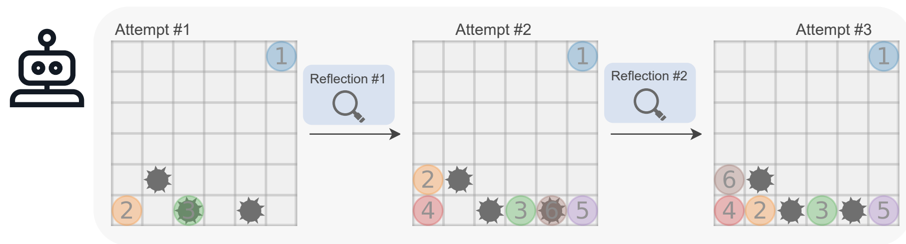
In our main experiment at Table 1, we follow previous work and evaluate the standard RL methods without access to the inter-episode memory. For comprehensive evaluation, we further evaluate the RL trained agents with access to the inter-episode memory (reflections and previous trajectories). The results of pass@3 are shown in Table 5. We observe that the inter-episode memory enhances the performance of RL trained agents on Sokoban (+3.8%) and MineSweeper (+5.3%), while degrades the performance on Webshop (-1.2%). Nevertheless, LAMER still substantially outperforms RL baselines across all the environments, demonstrating the advantage of the proposed method.

Table 5: Performance of RL baselines with access to inter-episode memory (pass@3).

Method	Sokoban	MineSweeper	Webshop
GiGPO (w/o memory)	44.1	55.1	75.2
GiGPO (w/ memory)	47.9	60.4	74.0
LAMER	**55.9**	**74.4**	**89.1**

E EXAMPLES

On Figure 6, we provide an example of trajectories and corresponding reflections produced by the agent when solving the MineSweeper game. Here each trajectory is represented by a sequence of clicks (numbered cells) on the board. The mines are not visible to the agent and will lead to failure of the game if clicked.



Reflection #1

Step-by-Step Reasoning
In the past attempt, the player revealed cells (6, 1) and (6, 3). The resulting state showed that row 6, column 1 is a 1, row 6, column 2 is a ?, row 6, column 3 is a *, and row 6, column 4 is a ?. This indicated that there is a mine at (6, 3), which is why the game ended in a loss.

Mistakes and Inefficiencies:
1. **Revealing (6, 1)**: This revealed a 1, indicating that there is exactly one mine in the 8 neighboring cells. However, since (6, 1) is adjacent to (6, 2) and (5, 1), this information is limited and not very helpful in identifying the exact location of the mine.
2. **Revealing (6, 3)**: This revealed a *, indicating that (6, 3) is a mine. This is a direct loss, but it also provides some information about the surrounding cells.
3. **Lack of strategic focus**: The player did not consider the implications of the numbers in row 4 and row 5, which are 1s and 2s, suggesting that there are nearby mines.

New Plan of Action
To improve the strategy, the player should focus on revealing cells that provide the most information without risking a direct reveal of a mine. Here's a revised plan:
1. **Reveal (5, 1)**: This cell is adjacent to (6, 1) and (4, 1). Revealing (5, 1) would help determine if there is a mine nearby, especially since (6, 1) is a 1, indicating that there is exactly one mine in the 8 neighboring cells.
2. **Reveal (5, 2)**: This cell is adjacent to (6, 2) and (4, 2). It would help determine if there is a mine near (6, 2), which is a ?.
3. **Reveal (6, 4)**: This cell is adjacent to (6, 3) (which is a mine) and (5, 4). Revealing (6, 4) would help determine if there is a mine nearby, especially since (6, 3) is a mine.

Reflection and Improved Plan
<remark>
In the past attempt, revealing (6, 1) and (6, 3) led to a loss because (6, 3) was a mine. To avoid this, the next step should be to reveal (5, 1) to check the surrounding cells without directly revealing a mine. After that, revealing (5, 2) would help determine if there is a mine near (6, 2). Finally, revealing (6, 4) would help identify any nearby mines adjacent to (6, 3). This approach would provide more information without risking a direct hit on a mine.
</remark>

Reflection #2

Step-by-Step Reasoning
In the past attempt, the player revealed the cells (5, 1), (6, 4), (6, 1), (6, 6), and (6, 5). The resulting state showed some progress, but the task was not completed successfully. Let's analyze what went wrong:

1. **Revealing (5, 1)**: This cell is adjacent to row 4, which has three 1s. Revealing (5, 1) would have revealed the surrounding cells, but since it was a 1, it indicated that there is at least one mine nearby. However, this didn't provide enough information to determine the exact location of the mines.
2. **Revealing (6, 4)**: This cell is adjacent to row 5, column 4, which is a 2. Revealing (6, 4) would have shown the number of mines around it, but since it was a 1, it indicated that there is at least one mine nearby.
3. **Revealing (6, 1)**: This cell is adjacent to row 5, column 1, which is a 1. Revealing (6, 1) would have shown the number of mines around it, but since it was a 1, it indicated that there is at least one mine nearby.
4. **Revealing (6, 6)**: This cell is adjacent to row 5, column 6, which is a 1. Revealing (6, 6) would have shown the number of mines around it, but since it was a 1, it indicated that there is at least one mine nearby.
5. **Revealing (6, 5)**: This cell is adjacent to row 5, column 5, which is a 1. Revealing (6, 5) would have shown the number of mines around it, but since it was a 1, it indicated that there is at least one mine nearby.

The main issue was that the player did not focus on revealing cells that would provide more direct information about the location of the mines. Instead, the player revealed cells that were adjacent to already revealed cells, leading to a lot of unnecessary information and not enough progress toward identifying the exact location of the mines.

New Plan of Action
To improve the strategy, the player should focus on revealing cells that are not adjacent to already revealed cells, and that are far enough away to avoid being affected by the auto-cascade effect. The goal is to identify the location of the mines by revealing cells that are not adjacent to any already revealed cells, and that are in areas where the numbers indicate a higher likelihood of a mine.

Here is a revised plan:
1. **Reveal (6, 2)**: This cell is not adjacent to any already revealed cells and is far enough away to avoid being affected by the auto-cascade effect. It is also adjacent to row 5, column 2, which is a 2, indicating that there is at least one mine nearby.
2. **Reveal (6, 3)**: This cell is not adjacent to any already revealed cells and is far enough away to avoid being affected by the auto-cascade effect. It is also adjacent to row 5, column 3, which is a 2, indicating that there is at least one mine nearby.
3. **Reveal (6, 1)**: This cell is adjacent to row 5, column 1, which is a 1, indicating that there is at least one mine nearby. However, since it is adjacent to row 6, column 2, which is a 1, it is important to reveal this cell to confirm the location of the mine.
4. **Reveal (6, 4)**: This cell is adjacent to row 5, column 4, which is a 2, indicating that there is at least one mine nearby. However, since it is adjacent to row 6, column 3, which is a 1, it is important to reveal this cell to confirm the location of the mine.
5. **Reveal (6, 5)**: This cell is adjacent to row 5, column 5, which is a 1, indicating that there is at least one mine nearby. However, since it is adjacent to row 6, column 4, which is a 1, it is important to reveal this cell to confirm the location of the mine.

By following this plan, the player can identify the location of the mines more effectively and avoid the mistakes made in the previous attempt.
<remark>
In the past attempt, the player revealed cells that were adjacent to already revealed cells, leading to a lot of unnecessary information and not enough progress toward identifying the exact location of the mines. To improve the strategy, the player should focus on revealing cells that are not adjacent to any already revealed cells, and that are far enough away to avoid being affected by the auto-cascade effect. The revised plan involves revealing cells such as (6, 2), (6, 3), (6, 1), (6, 4), and (6, 5) to identify the location of the mines more effectively.
</remark>

Figure 6: Example of trajectories and reflections produced by LaMer trained agents on the MineSweeper game.