SCALING IN-CONTEXT ONLINE LEARNING CAPABILITY OF LLMS VIA CROSS-EPISODE META-RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) achieve strong performance when all task-relevant information is available upfront, as in static prediction and instruction-following problems. However, many real-world decision-making tasks are inherently online: crucial information must be acquired through interaction, feedback is delayed, and effective behavior requires balancing information collection and exploitation over time. While in-context learning enables adaptation without weight updates, existing LLMs often struggle to reliably leverage in-context interaction experience in such settings. In this work, we show that this limitation can be addressed through training. We introduce ORBIT, a multi-task, multi-episode meta–reinforcement learning framework that trains LLMs to learn from interaction *in context*. After meta-training, Qwen3-8B, a relatively small open-source model, demonstrates substantially improved in-context online learning on entirely unseen environments: it matches or slightly exceeds GPT-4o while outperforming other baselines by a large margin on average. Beyond performance, we find that ORBIT exhibits adaptive exploration, revising its actions after failure without explicit reflection prompts. Scaling experiments further reveal consistent gains with model size, suggesting significant headroom for learn-at-inference-time decision-making agents.

1 INTRODUCTION

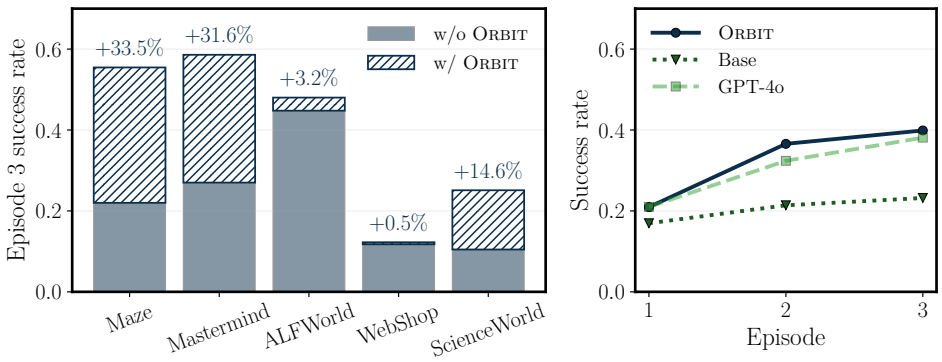


Figure 1: Left: ORBIT gains over the base model on episode-3 success rate for five unseen tasks (w/ ORBIT vs. w/o ORBIT). Right: Average success rate across all five tasks over episodes for ORBIT, Base model and GPT-4o.

Humans are remarkable learners. Although far from omniscient, we can often acquire new skills through a few rounds of trial and error: memory and reasoning allow us to discover effective strategies in a new game, interface, or problem domain. The ability to use interaction experience to improve behavior over time is therefore a fundamental aspect of intelligence.

In contrast, online learning remains a major challenge for large language models (LLMs). Despite strong in-distribution performance, LLMs typically require substantial post-training to adapt reliably

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

to new domains or skills; once deployed, they rarely improve through continued interaction in the way humans do. This "static-after-shipping" property limits their use as general-purpose agents that must operate in unfamiliar environments, recover from mistakes, and refine strategies from experience.

A promising alternative to continual weight updates is *in-context learning* (ICL), where adaptation occurs through the context window alone. While ICL enables LLMs to condition on demonstrations and feedback without parameter updates (Brown et al., 2020), autonomous agents require more than static input–output prediction: they must learn decision strategies from sequential experience. In partially observed interactive tasks, an agent must explore to acquire information, assign credit from delayed feedback, and exploit what it has learned. Empirically, even strong proprietary models such as GPT-4o struggle to reliably use in-context experience for online decision making (Fig. 1), suggesting that strong language modeling alone does not yield robust *in-context online learning*.

We study a practical regime in which an agent is given multiple attempts at the same unfamiliar task. Each attempt starts from a fresh initial state but follows the same underlying rules and environment structure, so early failures can provide information useful for later attempts. We call this setting *multi-episode in-context online learning*: adaptation must occur across repeated trials using only the interaction history stored in context, without parameter updates.

A key challenge is evaluating whether improvements reflect genuine in-context adaptation rather than task-specific policy learning. When agents are trained and evaluated within the same environment family, gains across episodes may partly reflect benchmark-specific knowledge acquired during training, such as action spaces, transition structure, or common solution patterns. For example, agents trained on *WebShop* (Yao et al., 2022) or *ALFWorld* (Shridhar et al., 2020) may improve on held-out tasks from the same benchmark without necessarily acquiring a transferable strategy for in-context online learning. Such evaluations are useful, but they do not fully isolate generalizable in-context adaptation.

This paper asks:

Can we endow LLMs with general-purpose in-context online learning capability?

We answer this question with ORBIT, a simple *multi-task, multi-episode meta–reinforcement learning* framework for training LLMs to adapt in context. ORBIT meta-trains a pretrained LLM across diverse partially observable decision-making environments and multiple episodes per task. The model is rewarded not merely for solving a single instance, but for learning to gather task-relevant information in early episodes and exploit it in later ones. Crucially, ORBIT is evaluated on entirely disjoint task families, providing a stricter test of transferable in-context reinforcement learning than evaluations on held-out instances from the same environment family. This contrasts with prior multi-episode agent-training work (Jiang et al., 2025; Yang et al., 2026), where training and evaluation are conducted within the same environment family and performance gains can be difficult to attribute uniquely to in-context adaptation. We also keep the framework minimal, avoiding external memory (Packer et al., 2023) or extensive reflection-style prompt engineering (Shinn et al., 2024), in order to isolate the effect of multi-episode meta-RL.

After meta-training, Qwen3-8B exhibits substantially improved in-context online learning on *completely unseen task families*. ORBIT matches or slightly exceeds GPT-4o while outperforming other baselines by a large margin on average (Figure 1 and Table 2). Beyond results, ORBIT shows adaptive exploration in *Maze*, revising behavior after failure without explicit reflection prompts. Scaling experiments further show consistent gains with model size, suggesting meaningful headroom for future progress. Overall, our results identify meta reinforcement learning as a promising path toward general-purpose agents that can learn effectively at inference time.

1.1 RELATED WORK

The In-Context Learning Phenomenon In-context learning (ICL) was popularized by the GPT-3 technical report (Brown et al., 2020), which demonstrated that large language models can follow task instructions from a small number of prompt demonstrations without parameter updates. Most subsequent work on ICL, and its mechanistic analyses (von Oswald et al., 2023; Ahn et al., 2023), focus on few-shot *instruction following* under static input–output formats (Christiano et al., 2017;

Ouyang et al., 2022; Rafailov et al., 2023); these settings do not require an agent to explore or adapt through ongoing interaction.

In-Context Reinforcement Learning With the advent of transformers (Vaswani et al., 2017), numerous works have trained sequence models as decision-making agents, most notably Decision Transformer (Chen et al., 2021). A growing body of work studies in-context reinforcement learning (ICRL) on classical RL benchmarks such as Atari, aiming to train sequence models that adapt online from interaction histories (Laskin et al., 2023; Lee et al., 2023; Lin et al., 2024; Raparthy et al., 2023). However, many of these approaches train models from scratch or from narrow trajectory distributions, which limits their ability to generalize to unseen tasks. In contrast, starting from a pretrained language model offers a strong semantic and reasoning prior, motivating our focus on general-purpose in-context RL built on top of pretrained LLMs.

In-Context Online and Reinforcement Learning in Language Models Recent studies have directly evaluated frontier LLMs as in-context online learners on canonical problems such as multi-armed bandits, often finding brittle or inconsistent exploration behavior in the absence of targeted training (Krishnamurthy et al., 2024; Nie et al., 2024; Felicioni et al., 2024; Park et al., 2025b; Zhang et al., 2025; Rahn et al., 2024; Monea et al., 2024; Sun et al., 2025). Motivated by these limitations, several recent works aim to improve the in-context RL capabilities of LLMs via post-training on self-generated interaction data. Iterative RMFT Park et al. (2025a) uses regret-based trajectory selection followed by iterative supervised fine-tuning, but is primarily evaluated in canonical multi-armed bandit settings. PAPRIKA (Tajwar et al., 2025) constructs preference signals from sampled trajectories for multi-turn post-training and similarly evaluates generalization to new tasks, but reports relatively weak performance on several environments (e.g., 2% success rates on Mastermind). Closest to our approach are LaMer (Jiang et al., 2025) and the concurrent work MAGE (Yang et al., 2026). LaMer's objective can be viewed as a discounted variant of ours, while MAGE assigns performance difference rewards for each episode. However, in both works, training and evaluation are conducted within the same environment family, so train and test tasks may share action spaces, transition dynamics, reward structures, and common solution patterns. As a result, observed improvements may reflect a mixture of test-time adaptation and environment-specific knowledge acquired during training, making it difficult to isolate genuine in-context learning from task-family-specific policy learning. In contrast, ORBIT is trained across a suite of partially observable tasks and evaluated on diverse, fully unseen environments. To the best of our knowledge, our work is the first to demonstrate substantial and transferable improvements in in-context reinforcement learning across diverse unseen task families.

2 IN-CONTEXT REINFORCEMENT LEARNING

We model each *task instance* as a finite-horizon episodic decision process

$$M = (\mathcal{S}, \mathcal{A}, P, r, \rho, H),$$

where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P(\cdot \mid s, a)$ is the transition kernel, $r(s, a)$ is the reward function, ρ is the initial-state distribution, and H is the (per-episode) horizon. All environments we consider are partially observable: task-defining parameters such as the maze layout or the hidden Mastermind code are part of the state but are never revealed to the agent, which receives only a textual observation of the current state. For notational simplicity we write the agent's input at each step as $s_{e,t}$, with the understanding that it is the observation emitted by the environment rather than the full latent state.

A *task* (e.g. *Maze*) is a family of instances that share the same action space, observation format, and rules; a task instance fixes the remaining parameters (the specific maze, start and goal positions, or the hidden code). An *episode* is a single attempt at an instance, starting from its initial state and lasting at most H steps or until a terminal state is reached. A *trajectory* $\tau^{(e)}$ is the sequence of observations, actions, and rewards collected in episode e.

In-context policies as online RL algorithms. We consider an LLM agent with parameters θ that selects actions using only the interaction transcript stored in its context window. In episode $e \in \{1, \ldots, T\}$ and within-episode timestep $t \in \{0, \ldots, H-1\}$, the agent observes $s_{e,t}$ and samples

Algorithm 1 Multi-Episode Interaction Protocol

Require: Task instance $M = (\mathcal{S}, \mathcal{A}, P, r, \rho, H)$, number of episodes T, policy π_θ
Ensure: Cross-episode interaction trace τ

```
Initialize empty history h ← ()
for e = 1 ... T do
    s ← env.reset()                                   (same instance, s ∼ ρ)
    for t = 0 ... H − 1 do
        Sample action a ∼ π_θ(· | h, s)
        Execute a and observe next state s′ and reward r = r(s, a)
        Update history h ← (h, s, a, r, s′)
        s ← s′
        if s is terminal then
            break
        end if
    end for
end for
return τ ≡ h
```

an action

$$a_{e,t} \sim \pi_\theta(\cdot \mid h_{e,t}),$$

where $h_{e,t}$ denotes the full history available in context up to that point. Concretely, we can write

$$h_{e,t} = \big(\tau^{(1)}, \ldots, \tau^{(e-1)}, s_{e,0}, a_{e,0}, r_{e,0}, \ldots, s_{e,t}\big),$$

i.e., the concatenation of all prior episodes and the partial trajectory of the current episode. The environment transitions as $s_{e,t+1} \sim P(\cdot \mid s_{e,t}, a_{e,t})$ and emits reward $r_{e,t} = r(s_{e,t}, a_{e,t})$. In this view, an online RL algorithm is simply a (possibly randomized) mapping from histories to action distributions; in-context reinforcement learning aims to realize such an algorithm through the LLM's forward pass, without any parameter updates at test time.

Multi-episode interaction protocol. To capture repeated trials of the *same* underlying task, we evaluate the agent on T episodes of interaction with a fixed task instance M. Each episode resets the environment to the initial state $s_{e,0} \sim \rho$ of the same instance while keeping (P, r) unchanged, and the agent retains the entire cross-episode transcript in its context window. We summarize the protocol in Algorithm 1. Let $\tau^{(e)}$ denote the trajectory collected in episode e:

$$\tau^{(e)} = (s_{e,0}, a_{e,0}, r_{e,0}, \ldots, s_{e,\ell_e}),$$

where $\ell_e \leq H$ allows for early termination. The full interaction trace across T episodes is the concatenation $\tau = (\tau^{(1)}, \ldots, \tau^{(T)})$.

Evaluation via in-context regret across episodes. Let the return of episode e be

$$G^{(e)} \;=\; \sum_{t=0}^{\ell_e - 1} r_{e,t}, \qquad \text{with } \ell_e \leq H,$$

and define the total return over T episodes as $\sum_{e=1}^{T} G^{(e)}$. Let

$$J^\star(M) \;=\; \max_{\pi} \; \mathbb{E}\left[\sum_{t=0}^{H-1} r(s_t, a_t)\right]$$

denote the optimal expected per-episode return on M (under the same initial-state distribution ρ and dynamics P). We define the *in-context regret* after T episodes as

$$\mathrm{Reg}_T(M) \;=\; T \cdot J^\star(M) \;-\; \mathbb{E}\left[\sum_{e=1}^{T} G^{(e)}\right],$$

where the expectation is over the environment randomness (initial states and transitions) and the agent's action sampling.

A strong in-context online learner is one whose regret grows slowly with T, indicating that it quickly extracts task-relevant information from early interaction and exploits it to achieve near-optimal performance in later trials. Minimizing $\mathrm{Reg}_T(M)$ induces an exploration–exploitation trade-off *across episodes*: actions in early episodes may sacrifice immediate reward to reduce uncertainty about the task, while the resulting information—retained in the context window—can be exploited to improve returns in subsequent episodes, all without any parameter updates.

3 ORBIT: ONLINE REINFORCEMENT-BASED IN-CONTEXT TRAINING

To train a in-context learning agent capable of solving diverse tasks on the fly, we train the agent on a collection of tasks (e.g., *Minesweeper* and *Blackjack*) and evaluate it on a disjoint set of unseen tasks (e.g., *Mastermind* and *Maze*). Since the two sets differ in dynamics, observation format, and solution strategy, success on the test tasks requires the agent to infer task-specific structure through in-context interaction.

During training, we sample a task M from the training task set $\mathcal{M}$. The agent then interacts with M for HT steps, generating a trajectory τ according to the multi-episode interaction protocol (Alg. 1) and accumulating reward $R(M;\tau)$. This training procedure can be viewed as a meta-level reinforcement learning problem, where the objective is to learn a non-Markovian meta-policy

$$\Pi : h_t \mapsto a_t,$$

implemented by the LLM acting purely in context. The meta-learning objective is to maximize the expected normalized cumulative reward, equivalently, to minimize regret over the task distribution,

$$\max_{\Pi} \mathbb{E}_{M\sim\mathcal{M},\Pi}[R(M;\tau)]\,.$$

3.1 REWARD DESIGN

To prevent reward hacking and ensure alignment with task completion, we define rewards at the trajectory level based on the number of successful task completions within a trajectory. For each task instance M, let $\mathcal{G}_M \subseteq \mathcal{S}$ denote the set of goal (terminal success) states, and define a success indicator for episode e as

$$\mathbb{I}_e^{(M)} = \mathbf{1}\big\{\exists\, t \le \ell_e : s_{e,t} \in \mathcal{G}_M\big\},$$

i.e., whether episode e reaches a goal state within its horizon.

We ignore any task-specific process rewards provided by the environment and instead use a unified binary completion reward. This choice avoids scale-induced imbalances across tasks: if different tasks exhibit heterogeneous reward magnitudes, tasks with larger intrinsic reward scales would dominate the optimization signal and contribute disproportionately large gradients. Using a standardized $0-1$ completion reward ensures balanced gradient contributions across tasks and focuses learning on successful task completion rather than exploiting task-specific reward shaping.

The trajectory-level reward is then defined as the success count

$$R(M;\tau) = \sum_{e=1}^{T} \mathbb{I}_e^{(M)},$$

which counts how many of the T episodes end in success. The corresponding meta-learning objective is to maximize the expected success count over the task distribution,

$$\max_{\Pi} \mathbb{E}_{M\sim\mathcal{M},\,\tau\sim\Pi}[R(M;\tau)]\,.$$

This objective encourages agents to reliably complete tasks rather than exploit dense or shaped reward signals, and naturally supports episodic environments in which multiple task completions may occur within a single trajectory.

Optimization. We optimize the meta-policy with Group Relative Policy Optimization (GRPO) (Shao et al., 2024): for each training instance we sample a group of K multi-episode trajectories, compute group-normalized advantages from their success counts $R(M;\tau)$, and apply PPO-style clipped policy-gradient updates over all generated tokens. No value function or per-step credit assignment is used, so the learning signal depends only on the relative number of successful episodes across the group. The full objective is given in Appendix F.

Table 1: Summary of training and test environments. All environments are partially observable and use $T = 3$ episodes. H denotes the maximum number of steps per episode.

Environment	Split	Task Description	H
RPS	Train	Infer opponent action distribution and adapt actions to maximize win rate	5
Minesweeper	Train	Infer hidden mines and reveal all safe cells under partial observability	8
Hangman	Train	Sequentially guess letters to uncover a hidden word with limited attempts	10
Wordle	Train	Infer a hidden word using structured feedback from each guess	10
Blackjack	Train	Infer deck composition to maximize expected return in a stochastic card game	4
Maze	Test	Explore an unknown map to reach a goal using local observations only	9
Mastermind	Test	Infer a hidden ordered sequence using structured feedback	3
ALFWorld	Test	Follow household instructions to complete multi-step object manipulation tasks	10
WebShop	Test	Search and navigate an website to purchase products matching user specifications	10
ScienceWorld	Test	Manipulate objects and conduct experiments to solve scientific reasoning tasks	15

4 EXPERIMENTS

4.1 ENVIRONMENTS

Table 1 summarizes all training and test environments along with concise task descriptions. All environments are partially observable in the sense that the environment parameters, such as the map of the maze, is unknown to the agent. Successful completion of the task requires the agent to balance exploration and exploitation. Detailed prompt designs and task specifications are provided in Appendix A and Appendix B.

All evaluations are conducted on five test environments: *Maze*, *Mastermind*, *ALFWorld* (Shridhar et al., 2020), *WebShop* (Yao et al., 2022), and *ScienceWorld* (Wang et al., 2022). These environments are entirely unseen during training. We use *Mastermind* from the GEM benchmark (Liu et al., 2025) and augment GEM with a new *Maze* environment. Among the test environments, *Mastermind* is structurally closest to the training tasks: like *Wordle* and *Hangman*, it involves guessing a fixed hidden target from feedback. Its Episode-1 success rate after training is nonetheless unchanged from the base model (Table 2), indicating that this similarity does not translate into a transferable zero-shot policy; the gains on *Mastermind* appear only after in-context interaction. The other four test environments share no comparable structure with any training task.

We intentionally keep the maximum number of steps per episode, H, relatively small. For example, in *Maze*, the shortest path from the starting position to the goal is constrained to be between 7 and 8 steps. This makes success difficult in early episodes, while requiring the agent in later episodes to exploit information gathered from prior interaction with little room for unnecessary detours. This evaluation protocol is designed to assess whether the model can generalize beyond the training tasks by using accumulated in-context information to adapt within a new environment.

4.2 EXPERIMENTAL SETUP

For the main comparison experiments and exploration analyses, we use Qwen3-8B as the base model (Yang et al., 2025). To study how in-context adaptation scales with model size, we additionally conduct a scaling study using Qwen3-{4B, 8B, 14B}. We further evaluate the generality of ORBIT beyond the Qwen3 family by running comparison experiments with GLM-4-9B-0414 (Glm et al., 2024); these results are reported in Appendix D. All base models used in our experiments have a maximum context length of 32,768 tokens. All experiments are implemented using the `rllm` training framework (Tan et al., 2025), except for LaMer. For LaMer, we use the official codebase released by the authors.

Baselines. We compare ORBIT against prompting-based baselines, a standard RL baseline, and a prior meta-RL method (Jiang et al., 2025). For the prompting-based baselines, we evaluate ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2024) using the same backbone model and interaction budget as ORBIT. For the RL baseline, we train a standard single-episode agent on the same training tasks as ORBIT. To ensure a fair comparison, we set the RL group size to three times the ORBIT group size, matching the total number of trajectories used in each optimization step. We also compare against

Table 2: Success rates across episodes on unseen environments. For each environment, we report success rates in Episodes 1–3. Results are averaged over 4 runs and rounded to two decimals. SE in each header bounds the standard error of a success rate in that environment (see Appendix G). Bold indicates the best average performance in each column before rounding.

	Maze (SE ≤ 0.02)			*Mastermind* (SE ≤ 0.02)			*ALFWorld* (SE ≤ 0.02)			*WebShop* (SE ≤ 0.03)			*ScienceWorld* (SE ≤ 0.02)		
Method	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3
ReAct	0.25	0.20	0.22	0.20	0.27	0.27	**0.26**	**0.40**	0.45	0.05	0.10	0.12	0.08	0.10	0.10
Reflexion	0.29	0.27	0.31	0.21	0.22	0.25	0.26	0.39	0.47	0.05	0.10	0.09	0.09	0.13	0.15
RL	**0.34**	0.44	0.42	**0.25**	0.37	0.26	0.23	0.34	0.43	0.04	0.10	**0.13**	0.13	0.19	0.21
LaMer	0.13	0.12	0.14	0.09	0.15	0.18	0.08	0.14	0.14	**0.13**	0.11	0.07	0.00	0.00	0.00
ORBIT	0.33	**0.50**	**0.55**	0.24	**0.56**	**0.59**	0.24	**0.40**	**0.48**	0.06	**0.13**	0.12	**0.18**	**0.24**	**0.25**

LaMer (Jiang et al., 2025), trained on the same multi-task suite as ORBIT with its default reflection-only context management (details and differences in Appendix E). All baselines are evaluated on the same held-out environments with the same episode budget as ORBIT.

Training setup. Unless otherwise specified, we use the same training configuration across models. During training, we sample with temperature 1.0 and top-$p = 1.0$. Trajectories that exceed the context-length limit are truncated and assigned zero reward. Each optimization step uses a batch of 64 task instances with $K = 4$ multi-episode trajectories per instance, i.e. 256 trajectories and 768 episodes per step. Models are trained for 100 optimization steps with a learning rate of 1×10^{-6}, using PPO-style clipped policy updates with asymmetric clipping bounds $(0.2, 0.28)$. Both entropy regularization and KL regularization are disabled. We enable the model's thinking mode during training to match the inference-time configuration.

Evaluation setup. For all experiments, including the scaling study in Section 4.5, we report the checkpoint obtained after 100 optimization steps. Model checkpoints are evaluated using the inference settings recommended for Qwen3 models, with temperature 0.6 and top-$p = 0.95$. Thinking mode is enabled during evaluation to match the training configuration.

4.3 ORBIT LEARNS IN CONTEXT

Table 2 reports, for each environment, success rates over three consecutive episodes within the same task instance. On *Maze*, *Mastermind*, and *ScienceWorld*, ORBIT shows the clearest in-context learning of all methods: it starts from an Episode-1 success rate comparable to the base model (ReAct) and improves substantially in Episodes 2 and 3, ending well above every baseline. On *ALFWorld* and *WebShop* the picture is more modest: all methods, including the untrained base model, improve across episodes, and ORBIT's Episode-3 success rate is on par with the best baseline rather than clearly above it.

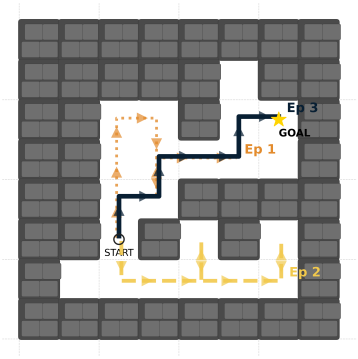
The comparison with the standard RL post-training baseline is the most informative one, since it is trained on the same task distribution and differs only in being optimized with single-episode completion rewards. In Episode 1 the two are indistinguishable, confirming that neither has acquired a transferable zero-shot policy for these environments. They separate in later episodes: the RL baseline's success rate plateaus or declines from Episode 2 to Episode 3 on *Maze* and *Mastermind*, whereas ORBIT continues to improve. Single-episode RL on the same tasks therefore does not, by itself, produce an agent that benefits from repeated attempts.

Because Episode-1 performance is flat across trained and untrained models, the Episode-3 gains on *Maze*, *Mastermind*, and *ScienceWorld* cannot be attributed to a stronger static policy learned during training; they arise from what the agent does with its own interaction history within a task instance. This is the central empirical claim of the paper, and it is the sense in which ORBIT learns in context.

Prompting baselines as a control for context length. The gains in later episodes could in principle come from simply seeing more of the environment in context rather than from meta-training. ReAct and Reflexion control for this: they use the same backbone, the same three-episode budget, and the same context format with the full cross-episode transcript retained, and differ from ORBIT only in not being meta-trained. Under this matched setting the base model improves only marginally from

Table 3: **Exploration under failure.** ΔStates (Ep2|F1) denotes the number of newly explored states in Episode 2 conditioned on failure in Episode 1, and ΔStates (Ep3|F1–2) denotes the number of newly explored states in Episode 3 conditioned on failure in both Episodes 1 and 2. Results are collected using Qwen3-8B.

Method	ΔStates (Ep2|F1)	ΔStates (Ep3|F1–2)
Base	1.64	0.94
RL	4.11	1.27
ORBIT	**4.69**	**1.48**



(a) Trajectory across three episodes. Episodes 1 and 2 explore suboptimal paths, while Episode 3 reaches the goal.

Reflect. In Episode 1, I went up three times, then right and down, but got stuck. In Episode 2, I went down first and explored the bottom corridor, but that also led to a loop.

Adapt. Previously, I went up again from here and failed to reach the goal. This time, I will explore right instead. right

Outcome. *[agent moves up, right, right, up, right].* **Congratulations!** You arrived at the goal.

(b) Reflection trace from Episode 3.

Figure 2: **Qualitative in-context adaptation in *Maze*.** ORBIT uses its interaction history to revise its exploration strategy after failed attempts. Panel (a) shows the three episode trajectories, and panel (b) highlights the corresponding reflection trace in Episode 3, where the agent identifies earlier failures, selects a different action, and reaches the goal.

Episode 1 to Episode 3 on *Maze*, *Mastermind*, and *ScienceWorld*, and an explicit reflection prompt does not close the gap, so a multi-episode context alone is not sufficient.

4.4 ORBIT LEARNS TO EXPLORE

Figure 2 provides qualitative evidence that ORBIT acquires an exploration strategy *through in-context adaptation* rather than relying on fixed heuristics or hand-engineered prompting. In the partially observable maze, the agent observes only local surroundings and must use the cross-episode transcript as its memory. After failing in Episodes 1–2, ORBIT enters Episode 3 and spontaneously performs behaviors commonly associated with deliberative exploration: it *reflects* on what went wrong, summarizes the relevant parts of the interaction history, and then chooses actions designed to gather new information instead of replaying the same trajectory.

Crucially, these behaviors are not prescribed by an explicit "reflect" instruction or an auxiliary module: they emerge from optimizing the multi-episode meta-RL objective alone. When faced with the same local observation, ORBIT selects actions that intentionally differ from earlier episodes (Figure 2(b)), indicating genuinely history-dependent decision making. This shift leads the agent to actively explore a previously unvisited route and successfully reach the goal (Fig. 2(a)).

Beyond the qualitative trace, Table 3 provides quantitative evidence that ORBIT changes its exploration behavior *conditioned on failure*. We measure how many *new* states are visited in later episodes given that the agent failed earlier: ΔStates (Ep2|F1) counts newly explored states in Episode 2 conditioned on failure in Episode 1, while ΔStates (Ep3|F1–2) counts newly explored states in Episode 3 conditioned on failure in both Episodes 1 and 2. Under this conditioning, an effective online learner should avoid repeating unproductive behaviors and instead actively expand coverage of the state

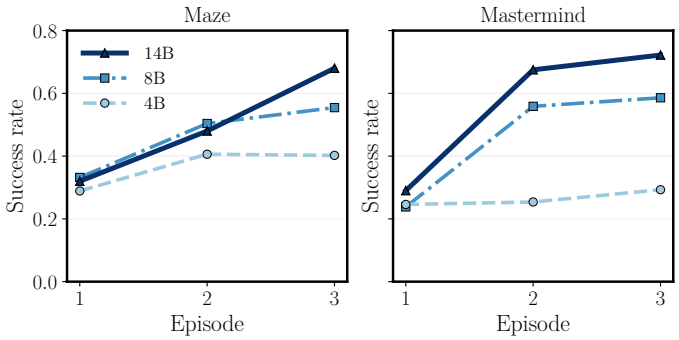


Figure 3: Success rate versus episode index for *Maze* (left) and *Mastermind* (right) when evaluating Qwen3-4B, 8B, 14B trained with ORBIT. All models are evaluated at the 100-step checkpoint.

space to reduce uncertainty. ORBIT consistently explores more new states than the single-episode RL baseline, suggesting that it learns to "try something different" after failure rather than cycling through the same local choices.

Taken together, the qualitative evidence (Fig. 2) and the quantitative analysis (Table 3) indicate that ORBIT learns an adaptive exploration policy in-context: it can perform reflection-like summarization and active exploration strategies on its own, using only the interaction history stored in the context window, and improves across episodes without any parameter updates at inference time.

4.5 ORBIT SCALES WITH MODEL SIZE

Figure 3 reveals a consistent benefit from scaling the Qwen3 backbone from 4B to 8B to 14B on the unseen *Maze* and *Mastermind* benchmarks, with the largest gains appearing in later episodes. In particular, Episode 3 improves the most with model size, followed by Episode 2, while Episode 1 shows only modest improvement (and can even drop for 14B).

This pattern is consistent with the interpretation that larger ORBIT models may allocate early interaction more toward information gathering (exploration) rather than immediate task completion (exploitation), and then capitalize on the accumulated cross-episode evidence in later attempts. Overall, scaling model capacity appears to amplify, rather than replace the multi-episode in-context learning dynamics induced by ORBIT.

5 DISCUSSION AND FUTURE WORK

This work primarily demonstrates the feasibility of using meta reinforcement learning as a mechanism for inducing generalizable ICRL capabilities, while leaving substantial room for further improvement. Importantly, we intentionally keep the framework as simple as possible, avoiding additional components such as external memory, retrieval-augmented generation, or extensive prompt engineering for summarization and reflection that have been explored in prior work, in order to isolate and highlight the contribution of **multi-episode meta learning** alone. Our experiments are limited to relatively short interaction horizons due to the limited context length; extending training to longer horizons, either with larger-context models or memory-augmented architectures, would enable richer temporal reasoning and more effective long-term credit assignment. In addition, we train on only five environments, and scaling both the number and diversity of training environments is a natural next step toward understanding empirical scaling laws governing generalization. There is also significant potential to improve the computational efficiency and stability of the training procedure through more advanced optimization and credit-assignment techniques.

Ethics Statement

This work studies training methods for LLM agents on simulated game and text-based environments. It does not involve human subjects, personal data, or sensitive content. All environments, base models, and training frameworks are publicly available open research assets (Appendix H.1). We do not foresee direct negative societal impacts specific to this method beyond those generally associated with more capable autonomous agents.

Reproducibility Statement

Training and evaluation configurations are described in Section 4, including sampling parameters, batch sizes, learning rates, and clipping bounds. Complete prompts for every environment are provided in Appendix A, task rules in Appendix B, and benchmark construction details (instance counts, seeds, and splits) in Appendix C. Hardware details and the exact GPT-4o snapshot used for comparison are given in Appendix H. All base models, environments, and the `rllm` framework are open source (Appendix H.1). Our training and evaluation code is included in the supplementary material.

AI Use Statement

We used LLM-based assistants in the following ways. *Writing:* editing for grammar, clarity, and concision throughout the paper. *Code:* assistance in writing and debugging experiment scripts, evaluation harnesses, and plotting code; all code was reviewed and tested by the authors. LLMs were not used to generate synthetic data, to design the method or the experimental protocol, to formulate or verify any mathematical claims, or to interpret results beyond the computations described above. The authors take full responsibility for all content.

References

Kwangjun Ahn, Xinyi Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.14165.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017. URL https://arxiv.org/abs/1706.03741.

Nicolò Felicioni, Lucas Maystre, Seth Ghiassian, and Kamil Ciosek. On the importance of uncertainty in decision-making with large language models. *arXiv preprint arXiv:2404.02649*, 2024. URL https://arxiv.org/abs/2404.02649.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-rl induces exploration in language agents. *arXiv preprint arXiv:2512.16848*, 2025.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? *arXiv preprint arXiv:2403.15371*, 2024. URL https://arxiv.org/abs/2403.15371.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.14215.

Jonathan N Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Neural Information Processing Systems*, 2023.

Licong Lin, Yu Bai, and Song Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=yN4Wv17ss3.

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, et al. Gem: A gym for agentic llms. *arXiv preprint arXiv:2510.01051*, 2025.

Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. Llms are in-context bandit reinforcement learners. *arXiv preprint arXiv:2410.05362*, 2024.

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. URL https://arxiv.org/abs/2410.06238.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. URL https://arxiv.org/abs/2203.02155.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Chanwoo Park, Ziyang Chen, Asuman Ozdaglar, and Kaiqing Zhang. Post-training LLMs as better decision-making agents: A regret-minimization approach. *arXiv preprint arXiv:2511.04393*, 2025a. URL https://arxiv.org/abs/2511.04393.

Chanwoo Park, Xiangyu Liu, Asuman Ozdaglar, and Kaiqing Zhang. Do llm agents have regret? a case study in online learning and games. In *International Conference on Learning Representations*, 2025b.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.18290.

Nate Rahn, Pierluca D'Oro, and Marc G Bellemare. Controlling large language model agents with entropic activation steering. *arXiv preprint arXiv:2406.00244*, 2024.

Sharath Chandra Raparthy, Eric Hambro, Robert Kirk, Mikael Henaff, and Roberta Raileanu. Generalization to new sequential decision making tasks with in-context learning. *arXiv preprint arXiv:2312.03801*, 2023.

Yi Ren and Danica J. Sutherland. Learning dynamics of LLM finetuning. *arXiv preprint arXiv:2407.10490*, 2024. doi: 10.48550/arXiv.2407.10490. URL https://arxiv.org/abs/2407.10490.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

Jiahang Sun, Zhiyong Wang, Runhan Yang, Chenjun Xiao, John Lui, and Zhongxiang Dai. Large language model-enhanced multi-armed bandits. *arXiv preprint arXiv:2502.01118*, 2025.

Fahim Tajwar, Yiding Jiang, Abitha Thankaraj, Sumaita Sadia Rahman, J. Zico Kolter, Jeff Schneider, and Ruslan Salakhutdinov. Training a generally curious agent. *arXiv preprint arXiv:2502.17543*, 2025. URL https://arxiv.org/abs/2502.17543.

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rllm: A framework for post-training language agents. https://pretty-radio-b75.notion.site/rLLM-A-Framework-for-Post-Training-Language-Agents-21b81902c146819db63cd98a54ba5f31, 2025. Notion Blog.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Johannes von Oswald, Eyvind Niklasson, Emanuele Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning (ICML)*, 2023.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

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.

Lu Yang, Zelai Xu, Minyang Xie, Jiaxuan Gao, Zhao Shok, Yu Wang, and Yi Wu. Mage: Meta-reinforcement learning for language agents toward strategic exploration and exploitation. *arXiv preprint arXiv:2603.03680*, 2026.

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

Ziyuan Zhang, Darcy Wang, Ningyuan Chen, Rodrigo Mansur, and Vahid Sarhangian. Comparing exploration-exploitation strategies of llms and humans: Insights from standard multi-armed bandit tasks. *arXiv preprint arXiv:2505.09901*, 2025.

A PROMPTS

System Prompt

You are solving the same task across multiple episodes with a fixed total step budget. Each episode resets the environment but keeps the task identical. Leverage information gathered from earlier episodes to succeed faster. Respond with actions inside \\box{} each turn.

Prompt for Minesweeper

You are playing the Minesweeper game. The objective of the game is to reveal all cells that do not contain mines. To make a move, you can either reveal a cell or place a flag on a suspected mine location using one of the following commands:
- 'reveal': Reveal the contents of a specific cell.
- 'flag': Place or remove a flag on a specific cell to mark it as a potential mine.
To submit your move, type the command followed by the row and column in \\box{} .
For example:
- \\box{reveal 4 4} to reveal the cell in Row 4, Column 4.
- \\box{flag 2 2} to place or remove a flag on the cell in Row 2, Column 2.
The current board layout is shown below. Cells that are unrevealed are represented by a dot ('.'), revealed numbers show the count of adjacent mines, and flagged cells are marked with an 'F'.
Use logic and deduction to avoid revealing cells with mines! Be mindful not to choose revealed or flagged cells.
Here is the current board layout:

```
    0  1  2  3  4
 0  .  .  .  .  .
 1  .  .  .  .  .
 2  .  .  .  .  .
 3  .  .  .  .  .
 4  .  .  .  .  .
```

Enter your guess.

Prompt for Hangman

You are playing Hangman. The objective of the game is to guess the 3-letter word by providing one letter guesses or the entire word.
The cells that need to be populated with letters are represented by '_'.
There are two ways you can answer. You can provide one letter guesses in the format of \\box{TOE}, or you can guess the entire word in the format of \\box{Y}.
If the given letter is in the word, it will be revealed in the grid.
If the given word is correct, you win.
As you play, the history of your choices will be appended below. Use the information to figure out the word and win.
Some rules:
1. You can only guess one letter/word at a time.
2. You have to win within 10 turns.
Here is the current state of the Hangman grid:

```
C00 C01 C02
 _   _   _
```

Enter your guess.

Prompt for Blackjack

You are an agent playing a simplified game of Blackjack against a dealer.

Game Objective:
Your goal is to choose actions that maximize your chance of winning against the dealer.
- The objective is to get a hand value as close to 21 as possible without exceeding 21.
- If your hand value exceeds 21, you bust and immediately lose.
- After you stand, the game ends and your hand is compared with the dealer's hand.

Card Values:
- Number cards (2–10): face value
- Face cards (J, Q, K): value 10
- Ace (A): value 1 or 11, chosen to give the highest possible hand value not exceeding 21

Dealer Rules:
- The dealer has exactly two cards.
- One dealer card is visible and the other is hidden.
- The dealer does not draw any additional cards.
- The dealer's hand value is computed using the same Ace rule as the player.

Initial Observation:
At the start of the episode, you observe:
1. Dealer's hand:
- One visible card and one hidden card
- Example: Dealer cards: [4, ?]

2. Your hand:
- A list of cards currently held
- Example: Your cards: [10, 6]

3. Deck:
- A list of remaining cards indexed by position
- Unknown card identities are hidden
- Example: Deck: [0: ?, 1: ?, 2: ?, 3: ?, 4: ?, ...]

Available Actions:
At each step, you must choose exactly one of the following actions:
- Hit: draw a specific card from the deck hit <card_index >
- Stand: stop drawing cards and end the game stand

Action Output Format:
You must output your action in exactly one line, using the following format:
- Hit example: \\box{hit 9}
- Stand example: \\box{stand}

Important constraints:
- Output only the boxed action.
- Do not include explanations, reasoning, or additional text.
- Do not output multiple actions.

Now Begin:
Given the current observation, decide your next action and output it in the required format.

Prompt for RockPaperScissors

You are playing a multi-turn Rock-Paper-Scissors game against an adversary.
In each episode, at every turn, the adversary's action is sampled from a fixed (hidden) distribution determined by the seed.
Your objective is to choose the action that maximizes the probability of winning against this hidden distribution each turn.
Now let's start the game. Output your action within \\box{...}.

Prompt for Wordle

You are playing Wordle.
You have to guess the secret 5-letter word within 10 turns.
After you enter your guess, I will say mark your guess as follows:
- G (green): correct letter in the correct position
- Y (yellow): letter exists in the word but in the wrong position
- X (wrong): letter is not in the word
After thinking, format your final answer inside \\box{...}, for example, CRANE.
As you play, the history of your guesses will be appended below. Use the information to complete the game before you run out of guesses.

Prompt for Mastermind

You are playing Mastermind.
You have to guess the secret 3 digit code within 3 turns.
The code consists of digits from 1 to 6 (inclusive).
Duplicate numbers are 'not allowed'.
After you enter your guess, I will say mark your guess with black and white pegs, where a black peg indicates a correct digit in the correct position, while a white peg indicates a correct digit in the wrong position.
After thinking, format your final answer inside \\box{...}, for example, \\box{1 4 6}.
As you play, the history of your guesses will be appended below. Use the information to complete the game before you run out of guesses.
Enter your first guess to start the game.

Prompt for Maze

You are a maze-solving agent. Your goal is to navigate from the START position to the GOAL position in the fewest turns possible. You are at the START position. {observation}
Output your next move from up/down/left/right within \\box{}.

Prompt for ScienceWorld

You are in a science lab. Your task is to perform a science experiment.

Task: {task_description}

{observation}

Available action types: open/close OBJ, activate/deactivate OBJ, connect OBJ to OBJ, disconnect OBJ, use OBJ on OBJ, look around, look at OBJ, look in OBJ, read OBJ, move OBJ to OBJ, pick up OBJ, put down OBJ, pour OBJ into OBJ, dunk OBJ into OBJ, mix OBJ, go to LOC, focus on OBJ, wait, inventory

Output your action inside \\box{} .

Prompt for ALFWorld

You are in an interactive household environment.
Your task is to: {task_description}

{observation}

Admissible actions: [{admissible_actions}]

Choose one admissible action and output it inside \\box{} .

Prompt for WebShop

You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to: {task_description}.
Prior to this step, you have already taken {step_count} step(s). Below are the history actions you took and the corresponding observation:

{observation}

Your admissible actions of the current situation are: [{admissible_actions}]
Now it's your turn to take one action for the current step. You should first reason step-by-step 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 \\box{} . For example, \\box{search[...]} or \\box{click[...]}.

B TASK RULES

We use ten environments for training or evaluation: *Minesweeper*, *Hangman*, *Rock–Paper–Scissors*, *Wordle*, *Blackjack*, *Mastermind*, *Maze*, *ALFWorld*, *WebShop*, and *ScienceWorld*. We summarize the rules of the first seven environments below. For the remaining three environments, *ALFWorld*(Shridhar et al., 2020), *WebShop*(Yao et al., 2022), and *ScienceWorld* (Wang et al., 2022), we refer readers to the original papers for detailed descriptions.

B.1 MINESWEEPER

Minesweeper is a grid-based puzzle game in which some cells contain hidden mines. Revealing a safe cell displays the number of mines in its eight neighboring cells. If a revealed safe cell has zero neighboring mines, all adjacent cells are automatically revealed, potentially triggering further expansions. The objective is to reveal all non-mine cells without revealing any mine; players may place flags to mark suspected mine locations. The game ends either when a mine is revealed (failure) or when all safe cells have been successfully uncovered (success).

B.2 HANGMAN

Hangman is a word-guessing game in which a target word is selected and initially concealed, with each letter represented by a placeholder. On each turn, the player guesses a single letter. If the guessed letter appears in the word, all corresponding positions are revealed; otherwise, the number of remaining attempts is reduced by one. The objective is to fully reveal the target word before the attempt limit is exhausted.

Example Suppose the hidden word is `APPLE`. A guess of `A` reveals `A _ _ _ _`. A subsequent guess of `P` reveals `A P P _ _`. An incorrect guess, such as `Z`, does not reveal any letters and consumes one attempt. The game continues until the word is fully revealed or the attempt limit is reached.

B.3 WORDLE

Wordle is a word deduction game in which a target word is selected and must be identified within a fixed number of attempts. On each turn, the player submits a word guess and receives feedback for each letter, indicating whether it is correct and in the correct position, correct but in an incorrect position, or absent from the target word. The objective is to deduce the target word before the attempt limit is reached.

Example Suppose the hidden word is `PLACE`. A guess of `ALIEN` receives feedback indicating that `L` is correct and in the correct positions, `A` and `E` are correct but in the wrong position, and `I` and `N` are absent from the word.

B.4 MASTERMIND

Mastermind is a code-breaking game in which a hidden sequence of numbers is randomly initialized. On each turn, the player submits a complete sequence as a guess and receives structured feedback. The number of black balls indicates how many numbers are correct and in the correct positions, while the number of white balls indicates how many numbers are correct but placed in incorrect positions. The objective is to identify the hidden sequence within a fixed maximum number of attempts.

Example Suppose the hidden code is (1, 2, 3). A guess of (1, 3, 4) receives feedback of one black ball (since 1 is in the correct position) and a white ball (since 3 is correct but not in the correct position).

B.5 MAZE

Maze is a randomly initialized grid-world environment consisting of a starting position, walls, and a goal. Each instance is constructed such that the shortest path from the start to the goal lies within a pre-specified length range. At each step, the player observes only the four adjacent grid cells and selects an action to move within the maze. The episode is successful if the player reaches the goal within a fixed maximum number of steps; otherwise, it is considered a failure.

B.6 ROCK-PAPER-SCISSORS

Rock–Paper–Scissors is a repeated game in which the player competes against an adversary with a fixed stochastic policy. An episode consists of multiple rounds. In each round, both the player and the adversary simultaneously choose one of three actions: rock, paper, or scissors. At the end of the episode, the player is declared the winner if they win more than half of the rounds; otherwise, the episode is considered a failure.

B.7 BLACKJACK

Blackjack is a traditional card game. We consider a simplified setting in which the dealer is dealt two cards, one hidden and one revealed. The player also starts with two cards and may repeatedly choose whether to draw an additional card from the deck or to stop. Once the player stops drawing, the sum of the player's cards is compared with the dealer's. The player wins if their sum is greater than the dealer's without exceeding 21; otherwise, the player loses. If the player's card sum exceeds 21 at any point, the episode immediately ends in a loss.

C ADDITIONAL BENCHMARK DETAILS

Maze. We evaluate on 256 randomly generated mazes with grid shapes drawn from $\{6 \times 6,\ 7 \times 6,\ 7 \times 7,\ 6 \times 7\}$ and shortest-path lengths between 7 and 8.

Mastermind. We evaluate on 256 instances of `game:Mastermind-v0-random` with code length 3, digits drawn from $\{1, \ldots, 6\}$, and no duplicate digits.

Table 4: Success rates across episodes on held-out evaluation environments on GLM-4-9B-0414. Results are averaged over 4 runs and rounded to two decimals; SE in each header is as in Table 2 (Appendix G). Bold indicates the best average performance in each column before rounding.

	Maze (SE $\leq$ 0.02)			*Mastermind* (SE $\leq$ 0.02)			*ALFWorld* (SE $\leq$ 0.02)			*WebShop* (SE $\leq$ 0.03)			*ScienceWorld* (SE $\leq$ 0.02)		
Method	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3	Ep1	Ep2	Ep3
ReAct	0.01	0.01	0.01	0.04	0.01	0.03	0.03	0.04	0.04	0.01	0.01	0.01	0.00	0.00	0.00
Reflexion	0.00	0.01	0.01	0.03	0.04	0.02	0.03	0.07	0.09	0.01	0.01	0.01	0.00	0.00	0.00
LaMer	0.02	0.03	0.02	0.05	0.04	0.03	0.00	0.01	0.00	**0.06**	**0.09**	**0.06**	0.00	0.00	0.00
RL	0.03	0.01	0.01	**0.05**	0.07	0.04	0.04	0.07	0.07	0.00	0.00	0.01	**0.01**	0.01	0.01
ORBIT	**0.04**	**0.04**	**0.05**	0.04	**0.09**	**0.11**	**0.07**	**0.10**	**0.11**	0.02	0.04	0.04	0.00	**0.01**	**0.02**

WebShop. We evaluate on the first 100 instances of the test split (`webshop:test-text`), selected deterministically by passing seeds $0, 1, \ldots, 99$ into the environment's session initializer.

ScienceWorld. We curate a subset of 6 task templates from the official `test` split, spanning three categories: classification (`find-animal`, `find-living-thing`, `find-plant`; 75 variations each, 225 total), electricity (`power-component` and `power-component-renewable-vs-nonrenewable-energy`; 5 variations each, 10 total), and biology (`identify-life-stages-2`; 4 variations). This yields 239 task instances in total.

ALFWorld. We evaluate on the full `eval_out_of_distribution` (`valid_unseen`) split, which contains 134 game files spanning the six standard ALFWorld task types.

D EXPERIMENTAL RESULTS ON GLM-4-9B-0414

We additionally validate ORBIT on GLM-4-9B-0414 (Glm et al., 2024). Unlike the Qwen-3 experiments, which are trained for 100 optimization steps, we train GLM-4-9B-0414 for 200 optimization steps, since we observe that its training-task performance plateaus after 100 steps and before 200 steps. All other training configurations are kept the same as those used for Qwen-3.

As shown in Table 4, ORBIT enables GLM-4-9B-0414 to generalize to completely unseen environments. Compared with prompting-based methods and the RL baseline, ORBIT achieves stronger overall performance across the evaluated environments. It also outperforms LaMer on all environments except WebShop. These results suggest that ORBIT is not specific to a single backbone and can generalize across different model architectures.

E COMPARISON WITH LAMER

Algorithmic differences. On the algorithmic side, the differences between ORBIT and LaMer (Jiang et al., 2025) are relatively limited. Both methods adopt multi-episode training, and LaMer's reward design can be viewed as a discounted variant of the success-count objective in ORBIT. The concrete differences are as follows.

- **Reflection.** LaMer applies an explicit reflection step after each failed episode, whereas ORBIT does not; any reflection-like behavior in ORBIT emerges from the meta-RL objective alone (Section 4.4).
- **Termination.** LaMer stops after the first successful episode, while ORBIT continues generating rollouts for the full episode budget even after a success, so that the success count over all T episodes is optimized.
- **Context management.** LaMer rebuilds the prompt from scratch at each episode. In its default mode, only the reflections from prior episodes are preserved; past observations, actions, and reasoning tokens are discarded. Other modes preserve past actions but not observations. ORBIT instead preserves the full interaction history across episodes by concatenating all observations, actions, and rewards, reducing information loss at the cost of a longer context.

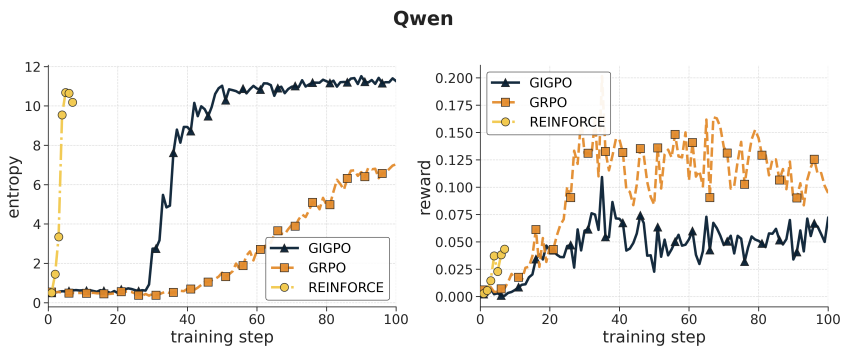


Figure 4: Entropy and reward during LaMer reproduction on GEM.

- **Advantage estimation.** LaMer uses GiGPO for advantage estimation, whereas ORBIT uses GRPO. The two are closely related (both are group-relative), and either framework could in principle be combined with a different post-training backbone.

Differences in evaluation. The main difference lies in the evaluation protocol. In LaMer, training and evaluation are conducted within the same environment family (e.g., sharing transition dynamics in ALFWorld). As a result, performance gains may reflect a combination of test-time adaptation and environment-specific knowledge acquired during training. In contrast, our evaluation is conducted on task families that are fully disjoint from those seen during training. Notably, in Table 2, the Episode-1 success rates of the RL and ORBIT models remain close to those of the untrained base model, which suggests that little task-specific knowledge is transferred from training into evaluation. The improvements that emerge only in later episodes therefore provide evidence that the gains come from adaptation based on interaction history rather than memorization of environment structure. In this sense, our contribution is not a new algorithmic paradigm, but a controlled empirical setting showing that a standard meta-RL formulation can elicit genuine in-context learning behavior.

Reproduction on GEM. We adapted LaMer-style training to our GEM multi-task setting. We first evaluated LaMer with its default GiGPO, and then with the commonly selected GRPO variant. Both variants exhibited unstable optimization: policy entropy increased sharply, as shown in Figure 4. Since GiGPO and GRPO use relative advantages, we initially suspected that the entropy growth might be caused by negative-advantage updates, which suppress lower-advantage samples and can push probability mass toward other actions. To isolate this effect, we further tried vanilla REINFORCE (Williams, 1992), which does not use group-relative negative advantages. However, REINFORCE also showed a rapid entropy increase and low reward. Due to resource constraints, we terminated the REINFORCE run early after observing this high-entropy trend.

The fact that REINFORCE still increases entropy suggests that LaMer's instability is not mainly driven by negative-advantage updates. Inspired by the learning-dynamics view of LLM finetuning (Ren & Sutherland, 2024), we hypothesize that the entropy growth is also related to the high similarity among reconstructed prompts within a rollout. Unlike ORBIT, which preserves the full cross-episode interaction transcript in context, LaMer reconstructs the prompt at each turn from a fixed template: the task description, summarized inter-episode memory, and the current observation/action context are inserted into predefined placeholders. This template-based reconstruction improves context efficiency, but it can make prompts from different steps or episodes look highly similar, even when they correspond to different underlying histories, actions, and outcomes.

This prompt similarity can create an entropy-increasing training signal. If a rollout of length H receives positive reward, training reinforces all H prompt-action pairs from that rollout. When these prompts are nearly identical but the reinforced actions differ, the model is encouraged to assign high probability to multiple distinct actions under similar contexts, which can increase policy entropy rather than sharpen the action distribution. This effect may be amplified in the GEM multi-task setting, where different environments produce different valid actions, delayed rewards, and reflection summaries under similar prompt templates. We therefore hypothesize that LaMer's template-based prompt reconstruction contributes to the entropy spikes observed in Figure 4.

We emphasize that this is a hypothesis rather than a definitive diagnosis. Nevertheless, the contrast with ORBIT suggests that preserving the full interaction history and optimizing a direct cross-episode success-count objective can provide a cleaner training signal in a general multi-task meta-RL setting. Among the tested LaMer variants, GRPO was noticeably more stable than GiGPO and REINFORCE in terms of entropy growth. Therefore, for the final LaMer baseline reported in our main experiments, we use the more stable GRPO variant for both the Qwen and GLM model series.

F Policy Optimization Details

For each training instance $M \sim \mathcal{M}$, we sample a group of K multi-episode trajectories $\{\tau^{(k)}\}_{k=1}^{K}$ from the current meta-policy Π_{old}, and assign each the success-count reward $R^{(k)} = R(M; \tau^{(k)})$ of Section 3. Following GRPO (Shao et al., 2024), the advantage is the group-normalized reward,

$$\hat{A}^{(k)} = \frac{R^{(k)} - \text{mean}\big(\{R^{(j)}\}_{j=1}^{K}\big)}{\text{std}\big(\{R^{(j)}\}_{j=1}^{K}\big)},$$

and is shared by every token of $\tau^{(k)}$. Let $o^{(k)} = (o_1^{(k)}, \ldots, o_{|o^{(k)}|}^{(k)})$ denote the sequence of tokens generated by the policy in $\tau^{(k)}$ (all action and thinking tokens across the T episodes; environment observations are excluded), and let $c_{<i}^{(k)}$ denote the context preceding token i, which contains the full cross-episode history up to that point. With the token-level importance ratio

$$r_i^{(k)} = \frac{\Pi\big(o_i^{(k)} \mid c_{<i}^{(k)}\big)}{\Pi_{\text{old}}\big(o_i^{(k)} \mid c_{<i}^{(k)}\big)},$$

the objective is

$$\mathcal{J}_{\text{GRPO}}(\Pi) = \mathbb{E}_{M\sim\mathcal{M},\, \{\tau^{(k)}\}\sim\Pi_{\text{old}}} \left[\frac{1}{K} \sum_{k=1}^{K} \frac{1}{|o^{(k)}|} \sum_{i=1}^{|o^{(k)}|} \min\Big(r_i^{(k)} \hat{A}^{(k)},\ \text{clip}\big(r_i^{(k)}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\big) \hat{A}^{(k)} \Big) \right],$$

with asymmetric clipping bounds $(\epsilon_{\text{low}}, \epsilon_{\text{high}}) = (0.2, 0.28)$. We disable the KL penalty to the reference policy and the entropy bonus. Because the advantage is constant across a trajectory, no per-step credit assignment is performed: the update depends only on the relative number of successful episodes within the group.

G Uncertainty Estimates

Each entry in Tables 2 and 4 is a success rate p averaged over 4 evaluation runs, each covering every task instance of the environment (256 for *Maze* and *Mastermind*, 134 for *ALFWorld*, 100 for *WebShop*, and 239 for *ScienceWorld*; Appendix C). Treating the $n = 4 \times$ (number of instances) episode outcomes as Bernoulli trials, the standard error of p is $\sqrt{p(1-p)/n}$, which is maximized at $p = 0.5$. The SE reported in each table header is this maximum, i.e. $0.5/\sqrt{n}$, rounded up to two decimals; the standard error of any individual entry is therefore at most the value shown, and differences larger than roughly twice this value are unlikely to arise from evaluation noise alone.

As an empirical check, across four independent evaluation reruns of the same checkpoint on *Maze* and *Mastermind*, for all three Qwen3 model sizes, the standard deviation of a success rate is at most 0.05 and typically 0.01–0.03. This captures decoding variance at inference time but not variance across training seeds, which we did not measure.

H Additional Experimental Details

8B and 4B Models are trained on a single node with 8 NVIDIA H100 GPUs and 14B models are trained on two nodes. For OpenAI model evaluation, we use `gpt-4o-2024-08-06` with temperature 0.6 and top-$p = 0.95$.

Table 5: Existing assets used in this work.

Asset	License / Terms
Qwen3-{4B, 8B, 14B}	Apache-2.0
GLM-4-9B-0414	MIT
ALFWorld	MIT
WebShop	MIT
ScienceWorld	Apache-2.0
GEM	Apache-2.0
`rllm`	Apache-2.0
LaMer	CC BY 4.0

H.1 Existing Assets and Licenses

We use existing open research assets, including base models, benchmark environments, and training codebases. The assets and their licenses or terms of use are summarized in Table 5.