EXPERIENTIAL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning has become the central approach for language models (LMs) to learn from environmental reward or feedback. In practice, the environmental feedback is usually sparse and delayed. Learning from such signals is challenging, as LMs must implicitly infer how observed failures should translate into behavioral changes for future iterations. We introduce *Experiential Reinforcement Learning* (ERL), a training paradigm that embeds an explicit experience–reflection–consolidation loop into the reinforcement learning process. Given a task, the model generates an initial attempt, receives environmental feedback, and produces a reflection that guides a refined second attempt, whose success is reinforced and internalized into the base policy. This process converts feedback into structured behavioral revision, improving exploration and stabilizing optimization while preserving gains at deployment without additional inference cost. Across sparse-reward control environments and agentic reasoning benchmarks, ERL consistently improves learning efficiency and final performance over strong reinforcement learning baselines, achieving gains of up to +81% in complex multi-step environments and up to +11% in tool-using reasoning tasks. These results suggest that integrating explicit self-reflection into policy training provides a practical mechanism for transforming feedback into durable behavioral improvement.

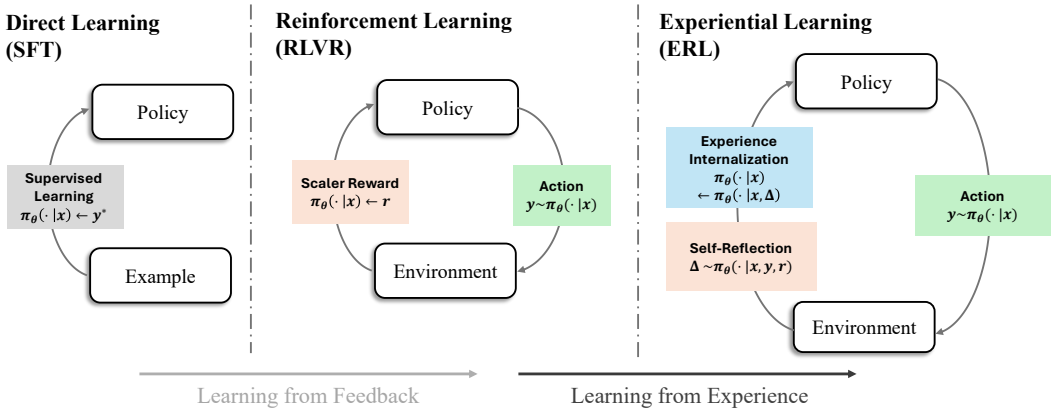

Figure 1: In *Experiential Reinforcement Learning* (ERL), instead of learning from feedback or outcome directly, an agent learns to (1) verbally reflect on its experience and observed outcome, and (2) internalize the reflections to induce behavioral changes in future iterations.

1 INTRODUCTION

Large language models are increasingly deployed as agents that must act, observe feedback, and adapt their behavior in environments with delayed rewards and partial information (Singh et al., 2025; Yang et al., 2025; Song et al., 2025a; Bai et al., 2026). Reinforcement learning offers a natural framework for improving such agents. The task environments typically provide feedback in the form of outcome reward after an agent generates the entire trajectory. In practice, training agents against such sparse and delayed outcome signals remains difficult, as models must implicitly infer how to translate observed failures into corrective behavior, a process that is often unstable and sample-inefficient

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.

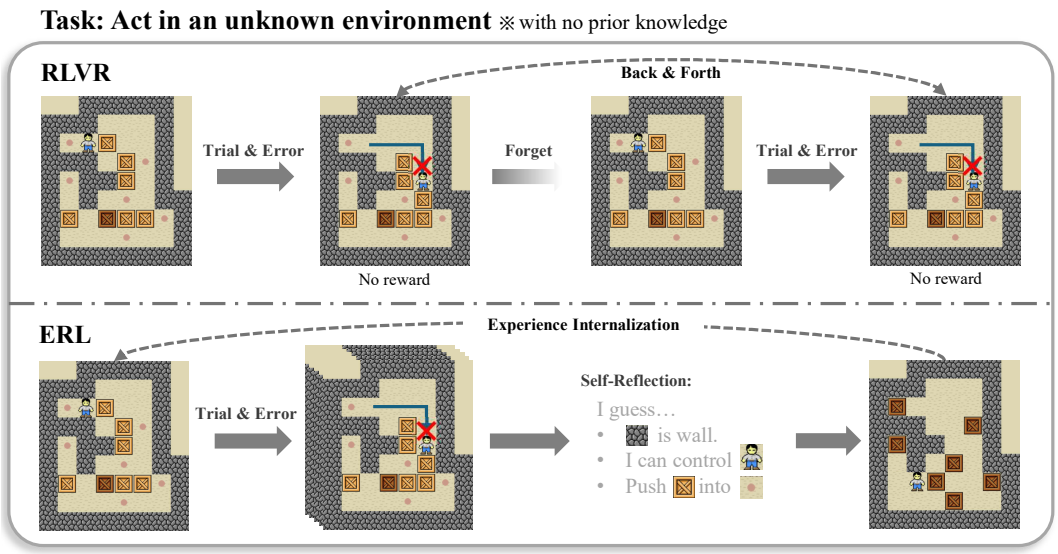

Figure 2: Conceptual comparison of learning dynamics in RLVR and *Experiential Reinforcement Learning* (ERL). RLVR relies on repeated trial-and-error driven by scalar rewards, leading to back-and-forth exploration without durable correction. ERL augments this process with an experience–reflection–consolidation loop that generates a revised attempt and internalizes successful corrections, enabling persistent behavioral improvement.

(Zhang et al., 2025; Shi et al., 2026). These challenges become more pronounced in agentic reasoning tasks, where multi-step decisions could amplify small errors and obscure credit assignment.

Humans address similar challenges through a process often described as *experiential learning*, in which effective adaptation arises from a cycle of experience, reflection, conceptualization, and experimentation (Kolb, 2014). After observing an outcome, a learner reflects on what occurred, forms revised internal models, and applies those revisions in subsequent attempts. This cycle transforms raw feedback into actionable behavioral corrections before those corrections are consolidated into future behavior. While language models have demonstrated reflection-like capabilities at inference time, standard reinforcement learning pipelines largely reduce feedback to scalar optimization signals, requiring policies to implicitly discover corrective structure through undirected exploration rather than explicit experiential revision.

This perspective highlights a progression in how language models learn from supervision and interaction, illustrated in Figures 1 and 2. In supervised fine-tuning (SFT), policies imitate fixed examples, enabling strong pattern reproduction but limiting behavior to the support of human-provided data. Reinforcement learning with verifiable rewards (RLVR) extends learning into interactive settings by optimizing scalar feedback, allowing agents to improve through trial-and-error; however, corrective structure must still be inferred implicitly from sparse or delayed rewards. As visualized in Figure 2, this can lead to repeated exploration without durable behavioral correction. A natural next step is to structure learning around experience, transforming feedback into intermediate reasoning that supports explicit revision and consolidation within each episode. Figure 1 conceptualizes this shift as moving from learning purely from feedback toward deliberate learning from experience.

In this work, we introduce *Experiential Reinforcement Learning* (ERL), a training paradigm that embeds an explicit experience–reflection–consolidation loop inside reinforcement learning. Instead of learning solely from outcome rewards, the model first produces an initial attempt, receives environment feedback, and generates a structured reflection describing how the attempt should be improved. This reflection conditions a refined second attempt, whose outcome is reinforced and internalized into the base policy. By converting feedback into intermediate reasoning signals, ERL enables the model to perform targeted behavioral correction before policy consolidation. Over time, these corrections become part of the policy itself, allowing improved behavior to persist even when reflection is absent at deployment. An overview of the algorithm is shown in Figure 3.

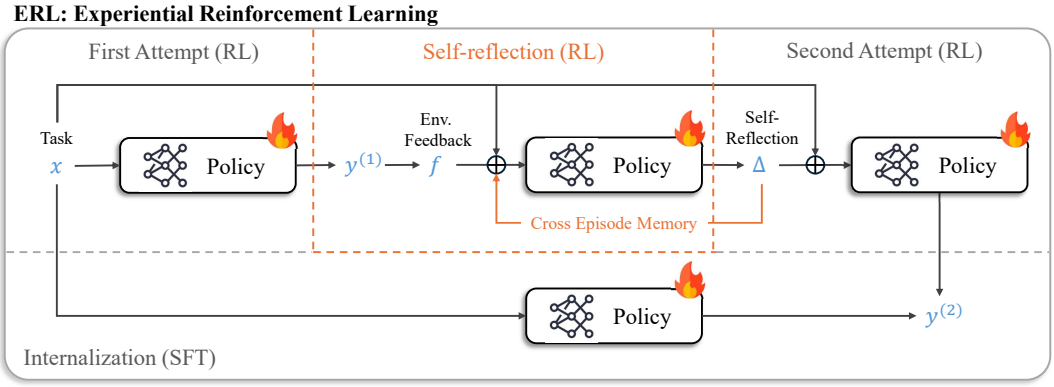

Figure 3: Overview of *Experiential Reinforcement Learning* (ERL). Given an input task x, the model first produces an initial attempt and receives environment feedback. The same model then generates a self-reflection conditioned on this attempt, which is used to guide a second attempt. Both attempts and reflections are optimized with reinforcement learning, while successful second attempts are internalized via self-distillation, so the model learns to reproduce improved behavior directly from the original input without self-reflection.

We evaluate ERL across sparse-reward control environments and agentic reasoning benchmarks spanning two model scales. ERL consistently outperforms RLVR in all six evaluated settings, achieving gains of up to +81% in Sokoban, +27% in FrozenLake, and up to +11% in HotpotQA. These results demonstrate that embedding structured experiential revision into training improves learning efficiency and produces stronger final policies across both control and reasoning tasks.

Contributions. Our main contributions are:

- We introduce *Experiential Reinforcement Learning* (ERL), a reinforcement learning paradigm that incorporates an explicit experience–reflection–consolidation loop, enabling models to transform environment feedback into structured behavioral corrections.
- We propose an internalization mechanism that consolidates reflection-driven improvements into the policy, preserving gains without requiring reflection at inference time.
- We demonstrate that experiential reinforcement learning improves training efficiency and final performance across agentic reasoning tasks.

2 Experiential Reinforcement Learning (ERL)

We introduce *Experiential Reinforcement Learning* (ERL), a training framework that enables a language model to iteratively improve its behavior through self-generated feedback and internalization. The key idea is to treat reflection as an intermediate reasoning signal that guides a refined second attempt, while reinforcement learning aligns both attempts with reward, and supervised distillation consolidates successful behaviors into the policy. An overview is shown in Figure 3, and the core training loop appears in Algorithm 1. A detailed implementation, including memory and gating logic, is provided in Appendix A.

Given an input task x, the model π_θ first produces an initial response

$$y^{(1)} \sim \pi_\theta(\cdot \mid x),$$

which is evaluated by the environment to produce textual feedback $f^{(1)}$ and reward $r^{(1)}$. Rather than immediately updating the policy, ERL optionally triggers a reflection-and-retry phase when the first attempt underperforms relative to a reward threshold τ. This selective retry mechanism focuses compute on trajectories that are most likely to benefit from revision while avoiding unnecessary refinement when performance is already sufficient. When triggered, the model generates a reflection

$$\Delta \sim \pi_\theta(\cdot \mid x, y^{(1)}, f^{(1)}, r^{(1)}, m),$$

Algorithm 1 Experiential Reinforcement Learning

Inputs: Language model π_θ; dataset of questions x; reward threshold τ; environment returning feedback f and reward r.
Initialize: reflection memory $m \leftarrow \emptyset$.
repeat
  Sample question x from the dataset.
  // First attempt
  Sample an answer $y^{(1)} \sim \pi_\theta(\cdot \mid x)$.
  Obtain environment feedback and reward $(f^{(1)}, r^{(1)})$.
  // Self-reflection
  Sample a reflection $\Delta \sim \pi_\theta(\cdot \mid x, y^{(1)}, f^{(1)}, r^{(1)}, m)$.
  // Second attempt
  Sample a refined answer $y^{(2)} \sim \pi_\theta(\cdot \mid x, \Delta)$.
  Obtain environment feedback and reward $(f^{(2)}, r^{(2)})$.
  Set reflection reward $\tilde{r} \leftarrow r^{(2)}$.
  Store reflection $m \leftarrow \Delta$ if $r^{(2)} > \tau$.
  // RL update
  Update θ via $\mathcal{L}_{\text{policy}}(\theta)$ over the first attempt, reflection, and second attempt.
  // Internalization
  Update θ via $\mathcal{L}_{\text{distill}}(\theta)$ to internalize reflection, training π_θ to produce $y^{(2)}$ from x only.
until converged

which serves as self-guidance describing how the initial attempt can be improved. Here, m denotes a cross-episode reflection memory that persists successful corrective patterns discovered during training. This memory provides contextual priors that help stabilize reflection generation and encourage reuse of previously effective strategies. The model then produces a refined response

$$y^{(2)} \sim \pi_\theta(\cdot \mid x, \Delta),$$

and receives new feedback $(f^{(2)}, r^{(2)})$. Reflections that lead to sufficiently improved outcomes are stored back into memory,

$$m \leftarrow \Delta \quad \text{if} \quad r^{(2)} > \tau,$$

allowing corrective knowledge to accumulate across training episodes. The reflection is assigned reward $\tilde{r} = r^{(2)}$, encouraging reflections that lead to improved downstream performance.

Both attempts and reflections are optimized using a reinforcement learning objective

$$\mathcal{L}_{\text{policy}}(\theta) = -\mathbb{E}[A \log \pi_\theta(y \mid x, \cdot)],$$

where y denotes model outputs arising from the first attempt, reflection, or second attempt, and the conditioning context corresponds to the inputs specified in Algorithm 1. The advantage estimate A is computed from the associated rewards.

While reflection and environment feedback provide strong training signals, such supervision is typically unavailable at deployment time, where the model must operate in a zero-shot setting. We therefore introduce an internalization step that converts reflection-guided improvements into persistent policy behavior. The goal is to make the model remember corrections discovered during training and avoid repeating the same mistakes when feedback is absent. We implement internalization via selective distillation: we supervise the model to imitate only successful second attempts while removing reflection context from the input. Concretely, given a training example x, we generate a refined response $y^{(2)}$ and reward $r^{(2)}$, and optimize

$$\mathcal{L}_{\text{distill}}(\theta) = -\mathbb{E}\Big[\mathbb{I}\Big(r^{(2)} > \tau\Big) \log \pi_\theta\Big(y^{(2)} \mid x\Big)\Big],$$

where $\mathbb{I}(\cdot)$ is the indicator function. This trains π_θ to reproduce improved behavior from the original input x alone (no reflection), ensuring that lessons learned through feedback and self-reflection persist at test time.

By alternating between reinforcement learning, selective reflection, and distillation, ERL bootstraps self-improvement: reflections guide higher-quality retries, memory preserves effective corrective structure, reinforcement learning aligns behavior with reward, and distillation internalizes gains into the core model. Over time, this interaction stabilizes training, concentrates exploration on failure cases, and reduces dependence on explicit reflection at inference.

3 EXPERIMENT

3.1 TASK

We evaluate *Experiential Reinforcement Learning* (ERL) on three agentic reasoning tasks against standard RLVR: Frozen Lake, Sokoban, and HotpotQA (Yang et al., 2018). Detailed environment descriptions are provided in Appendix C.

For Frozen Lake and Sokoban, we configure the environments with sparse terminal rewards following Wang et al. (2025) and Guertler et al. (2025). The agent receives reward only at episode completion: a reward of +1 is assigned for successfully achieving the objective and 0 otherwise. Crucially, *we do not provide explicit game rules or environment dynamics*. The model must infer task structure purely through interaction, with access limited to the available action set. This evaluation design is inspired by prior work on learning from experience, where the goal is to measure an agent's ability to acquire task understanding through trial-and-error rather than relying on human-authored priors embedded in pretraining. The combination of sparse rewards and unknown dynamics therefore creates a challenging setting that emphasizes reasoning, planning, and experiential learning.

HotpotQA is adapted into an agentic multi-hop question-answering task following Search-R1 (Jin et al., 2025). Given a question, the model performs iterative tool-assisted retrieval before producing a final answer. To maintain consistency with the experiential learning setup, we provide only a default system prompt describing available tools, without additional task-specific guidance. Correctness is evaluated using token-level F1 against ground-truth answers. The reward function assigns 1.0 for exact matches, a proportional reward for partial matches with F1 score ≥ 0.3, and 0 otherwise.

3.2 MODELS AND BASELINES

In our experiments, we train Olmo-3-7B-Instruct (Olmo et al., 2025) and Qwen3-4B-Instruct-2507 (Yang et al., 2025) using both standard RLVR and our proposed ERL paradigm, with GRPO (Shao et al., 2024) serving as the underlying policy-gradient optimizer in all cases. To ensure stable training, we adopt common reinforcement learning techniques such as clipping, KL regularization, and importance sampling. Notably, the internalization stage in ERL naturally involves off-policy data, which can introduce additional instability. We therefore apply the same stabilization techniques during this phase to maintain consistent optimization behavior. Additionally, because ERL requires two attempts per task along with an additional reflection step, we allocate 10 rollouts per task for RLVR and half as many per task per attempt for ERL to equalize the training compute per task across methods. Full hyperparameters and implementation details are provided in Appendix D.

4 RESULT AND DISCUSSION

We evaluate *Experiential Reinforcement Learning* (ERL) against standard RLVR across three environments spanning sparse-reward control (FrozenLake, Sokoban) and agentic reasoning (HotpotQA). Table 1 summarizes the final performance, while Figures 5–6 visualize the performance and learning dynamics. All curves are smoothed with a trailing moving average over 5 points. The same smoothing procedure is applied to all figures unless otherwise noted.

4.1 PERFORMANCE ACROSS TASKS

ERL consistently improves final evaluation performance over RLVR across all tasks and both model backbones. As shown in Table 1 and Figure 5, experiential training yields gains ranging from moderate improvements on HotpotQA to substantial improvements on Sokoban and FrozenLake.

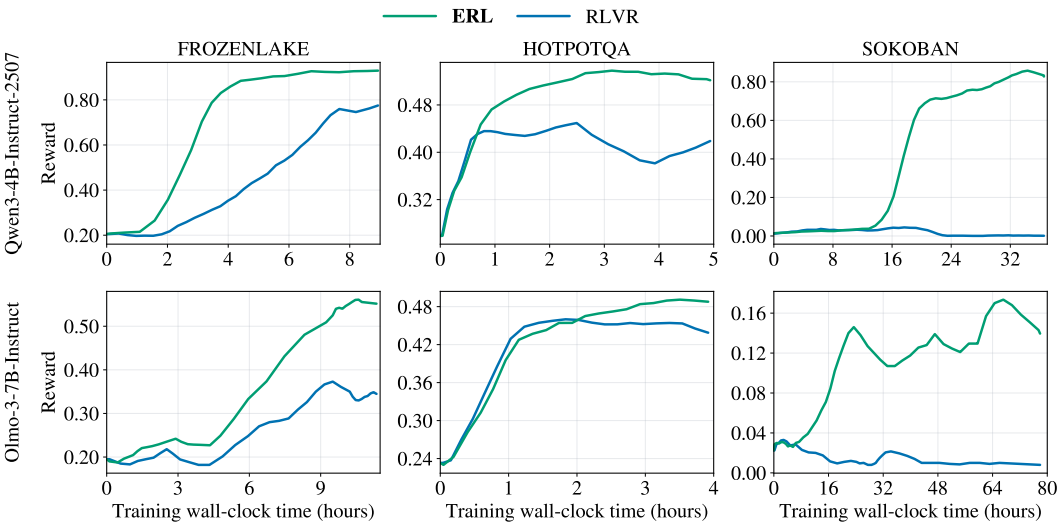

Figure 4: Validation reward trajectories versus training wall-clock time on FrozenLake, HotpotQA, and Sokoban for Qwen3-4B-Instruct-2507 and Olmo-3-7B-Instruct. ERL consistently achieves higher reward and faster improvement than RLVR across tasks and models.

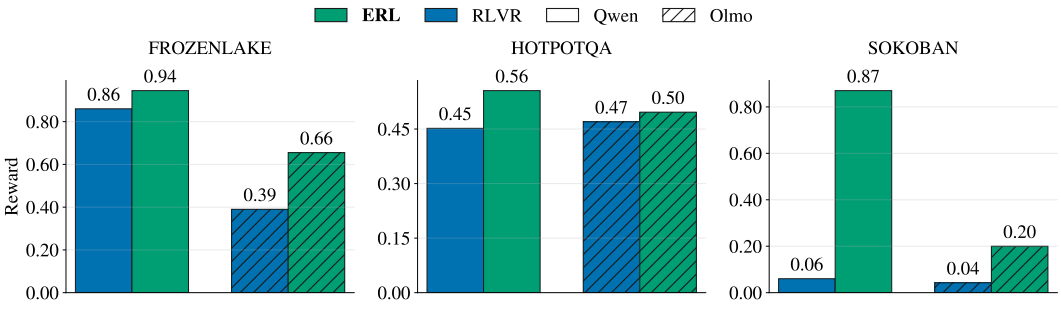

Figure 5: Final evaluation reward on FrozenLake, HotpotQA, and Sokoban. ERL consistently outperforms RLVR for both Qwen3-4B-Instruct-2507 and Olmo-3-7B-Instruct.

The largest effect occurs in Sokoban, where Qwen3-4B-Instruct improves from 0.06 to 0.87 and Olmo3-7B-Instruct from 0.04 to 0.20. Sokoban requires long-horizon planning and recovery from compounding errors, making performance sensitive to how well the agent reasons about environment dynamics. Similarly, FrozenLake demands that the agent infer symbol semantics, action consequences, and terminal conditions purely through interaction under sparse rewards. Importantly, as described in Section 3, unlike many prior evaluation setups that provide explicit rules or environment structure, our environments expose only observations and action interfaces; the agent must infer task dynamics through trial-and-error. This design emphasizes learning from experience rather than relying on pre-specified priors, making structured revision particularly valuable. In these settings, the experience–reflection–consolidation loop enables the model to analyze failures, revise strategies, and internalize corrective behavior within each episode, producing large improvements in exploration efficiency and policy quality.

HotpotQA shows smaller but reliable gains. Compared to the grid-based control environments, HotpotQA presents a more homogeneous interaction pattern centered on repeated tool invocation and answer synthesis, with denser reward and fewer latent dynamics to infer. Because RLVR already receives relatively informative gradients in this regime, the additional benefit of structured experiential revision is reduced. This contrast suggests that ERL yields the greatest advantage in environments where learning requires substantial reasoning about unknown dynamics and long-horizon consequences, rather than primarily optimizing over a stable interaction loop.

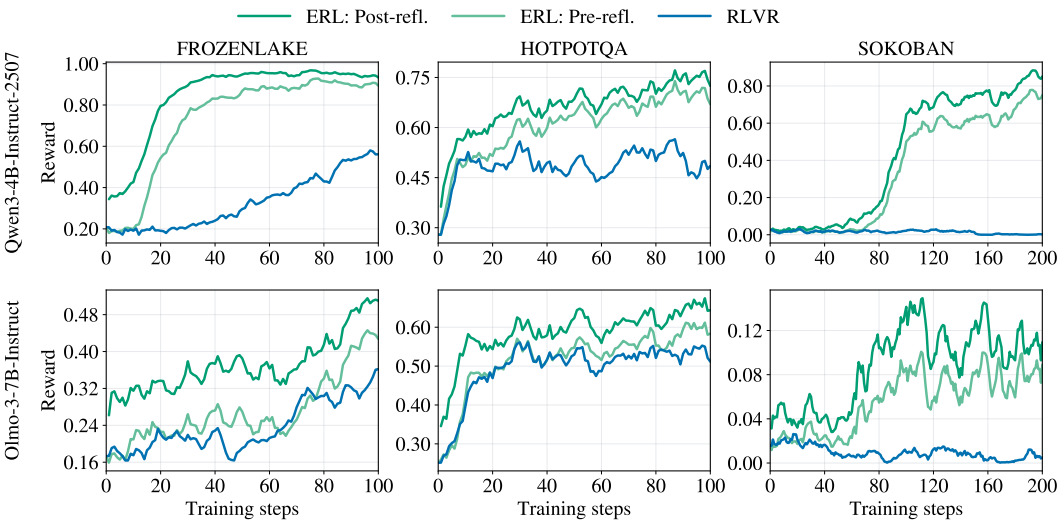

Figure 6: Training reward trajectories for Qwen3-4B-Instruct-2507 and Olmo-3-7B-Instruct comparing RLVR with ERL before and after reflection. Post-reflection trajectories consistently achieve higher reward than both RLVR and pre-reflection trajectories.

4.2 Learning Efficiency and Optimization Dynamics

Figure 4 compares validation reward against wall-clock training time. Across tasks and models, ERL reaches higher reward earlier and maintains a persistent margin over RLVR. This acceleration is especially pronounced in FrozenLake and Sokoban, where RLVR progresses gradually while ERL rapidly approaches high-reward behavior. These dynamics suggest that reflection introduces an intermediate corrective signal that reshapes exploration. Instead of relying solely on terminal reward propagation, the model conditions on feedback and self-generated critique to revise its behavior. This concentrates training updates on trajectories that are already partially aligned with the objective, reducing inefficient exploration. Even in HotpotQA, where rewards are denser and the environment is comparatively simpler, ERL maintains a consistent performance advantage over RLVR. Across environments, these results indicate that ERL achieves higher final reward while improving learning efficiency, demonstrating that structured experiential revision leads to faster and more effective policy improvement.

4.3 Mechanistic Role of Reflection

Figure 6 shows training reward trajectories for ERL before and after the reflection step, alongside RLVR. Across environments and models, post-reflection trajectories consistently achieve higher training reward than pre-reflection trajectories and also exceed RLVR. This comparison highlights the immediate within-episode effect of reflection. After observing feedback from the first attempt, the model generates a structured revision that guides a second attempt with improved actions. As qualitatively illustrated in Appendix B, these revisions take the form of explicit rule induction, strategy formation, and constraint enforcement, which translate feedback into actionable behavioral updates. The resulting gain in training reward indicates that reflection produces actionable corrections within the same episode, rather than only shaping behavior over long horizons. The sustained separation between pre- and post-reflection curves throughout training suggests that reflection serves as a systematic revision mechanism. By converting observed outcomes into targeted adjustments, it improves the quality of second attempts, which are subsequently reinforced and contribute to longer-term policy improvement.

4.4 Ablation Study: Memory and Reflection Mechanisms

To understand how structured reflection and cross-episode memory contribute to performance, we conduct ablation studies across tasks and models. The quantitative results are reported in Table 1,

Task	RLVR	ERL	ERL w/o Mem.	ERL w/o Refl.
Qwen3-4B-Instruct-2507				
FrozenLake	0.86	0.94	0.86 (-0.08)	0.60 (-0.34)
HotpotQA	0.45	0.56	0.56 (-0.00)	0.48 (-0.08)
Sokoban	0.06	0.87	0.87 (-0.00)	0.59 (-0.28)
Olmo3-7B-Instruct				
FrozenLake	0.39	0.66	0.64 (-0.02)	0.54 (-0.12)
HotpotQA	0.47	0.50	0.47 (-0.03)	0.46 (-0.04)
Sokoban	0.04	0.20	0.24 (+0.04)	0.06 (-0.14)

Table 1: Final evaluation reward on FrozenLake, HotpotQA, and Sokoban. ERL performance is compared against ablation variants, with highlighted drops showing the performance degradation relative to ERL when removing memory reuse (w/o Mem.) or structured reflection (w/o Refl.).

and representative learning dynamics for FrozenLake with Qwen3-4B-Instruct-2507 are shown in Figure 7. These experiments isolate individual components of ERL while keeping the overall training setup fixed.

The **no-memory** variant disables cross-episode reflection storage. Reflections are still generated and used to guide the second attempt within each episode, but they are not retained for reuse in future episodes. As a result, corrective signals remain local to individual trajectories rather than accumulating into persistent behavioral priors.

The **no-reflection** variant preserves the two-attempt interaction structure but removes explicit structured reflection. Instead, the model receives the full first-attempt interaction history together with a generic instruction encouraging improvement. This design tests whether contextual reuse alone can replicate the benefits of structured reflective reasoning. The prompt template used in this setting is shown in Table 15 (Appendix).

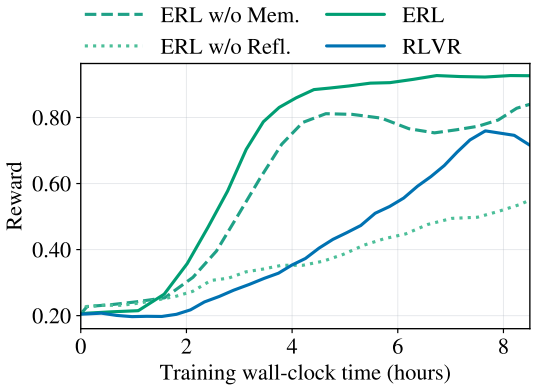

Figure 7: Ablation study on Qwen3-4B-Instruct-2507 in FrozenLake. We compare full ERL with two variants: removing memory reuse (w/o Mem.) or structured reflection (w/o Refl.).

The results in Table 1 show a consistent ordering across most tasks and models: full ERL achieves the strongest performance, followed by the no-memory variant, while the no-reflection variant exhibits the largest degradation in most settings. Figure 7 further illustrates that removing memory slows convergence, whereas removing reflection substantially reduces both learning speed and final reward. These findings support the core design intuition of ERL: reflection generates actionable behavioral corrections, and memory propagates those corrections across episodes to enable cumulative refinement.

At the same time, Table 1 reveals an important caveat. In the Olmo3-7B-Instruct Sokoban setting, the no-memory variant slightly outperforms full ERL. This suggests that when a model's self-reflective ability is limited, or when the environment is complex and stochastic, persistent memory may propagate early inaccurate reflections, making recovery more difficult. In such cases, disabling cross-episode memory can mitigate the accumulation of erroneous priors. Nevertheless, across the broad set of tasks and models evaluated, ERL consistently delivers the strongest overall performance, demonstrating that structured reflection combined with persistent memory is highly effective in most practical settings.

5 Related Work

Reinforcement Learning for LLMs. Reinforcement learning has become a central approach for improving large language models. Early work focused on reinforcement learning from human feedback (RLHF) to align model behavior with human preferences and conversational objectives (Ouyang et al., 2022; Christiano et al., 2023; Shi et al., 2024; 2025). More recent efforts extend RL to enhance mathematical reasoning, where verifiable or programmatic rewards derived from executable checks or formal answer verification provide structured supervision for reasoning and solution construction (OpenAI et al., 2024; Guo et al., 2025; Song et al., 2025b; Shi et al., 2026). In parallel, research on tool-using and agentic LLMs treats the model as a policy that interacts with external environments, alternating between actions and observations under task-dependent rewards to solve multi-step problems (Yao et al., 2023; Jin et al., 2025; Bai et al., 2026; Jiang et al., 2026). Despite their different goals, these approaches primarily treat environment feedback as a scalar optimization signal propagated through policy gradients, requiring the model to implicitly infer corrective structure through exploration. In contrast, our ERL paradigm introduces an explicit experience-reflection-consolidation loop that transforms environment feedback into structured behavioral revision before internalizing improvements into the base policy.

Learning from Experience. A growing body of work argues that the next scaling regime for AI will come not from more static human text, but from agents generating ever-richer data through interaction-i.e., learning predominantly from experience. Silver & Sutton (2025) emphasizes that continual, agent-generated data streams and long-horizon decision-making as the route beyond imitation of human corpora. This motivates algorithmic mechanisms that convert failures into usable learning signal rather than relying on rare successes. In classic reinforcement learning, Andrychowicz et al. (2018) addresses sparse rewards by relabeling goals so that failed trajectories can still provide informative updates, substantially improving sample efficiency in goal-conditioned tasks. In the LLM-agent setting, Zhang et al. (2025) similarly targets the gap between imitation and full RL by training agents on their own interaction traces even when explicit rewards are unavailable, using the agent's generated future states as supervision and including self-reflection as a way to learn from suboptimal actions. Meanwhile, inference-time reflection methods demonstrate that LLMs can critique and revise their own outputs to improve success (Zelikman et al., 2022; Madaan et al., 2023; Shinn et al., 2023), but typically require reflection or memory at deployment. Concurrent research explores integrating feedback-conditioned improvement directly into training. Hübotter et al. (2026); Song et al. (2026); Ye et al. (2026) formalize RL with textual feedback by distilling a feedback-conditioned teacher policy into a student policy. ERL is aligned with this direction but emphasizes explicit self-reflection as an intermediate reasoning step embedded inside the RL trajectory, where an initial attempt is followed by reflection and a refined retry. Coupled with selective internalization and cross-episode memory, this design treats reflection as a structured credit-assignment mechanism that transforms raw experience into durable behavioral improvement without requiring reflection at inference time.

6 Conclusion

In this work, we presented Experiential Reinforcement Learning (ERL), a training paradigm that incorporates an explicit experience–reflection–consolidation stage into the reinforcement learning loop to convert environment feedback into structured behavioral correction. By pairing reflection-guided revision with selective internalization, ERL enables models to learn corrective strategies during training and consolidate them into a deployable policy that operates without reflection at inference time. Across sparse-reward control and agentic reasoning tasks, ERL improves learning efficiency, stabilizes optimization, and produces stronger final policies relative to standard reinforcement learning baselines. These results demonstrate that embedding structured experiential revision directly into the training process provides an effective mechanism for translating feedback into durable behavioral improvement. Looking forward, this work suggests a path toward reinforcement learning systems that are fundamentally grounded in experience, where explicit reflection and consolidation become core primitives for building agents that continually learn, adapt, and improve from their own interactions.

AI USE STATEMENT

In this work, we used generative AI tools to polish grammar and improve the clarity of the writing, assist with debugging code, and support the literature review process. We did not use generative AI tools for generating research ideas, performing data analysis, interpreting results, or generating scientific claims, and other disclosure tasks are not applicable to this work. We reviewed all AI-assisted work: language edits were manually checked to ensure they preserved the intended meaning and scientific accuracy, literature-review assistance was verified against the original sources and relevant literature, and AI-assisted code debugging was reviewed and tested by the authors to verify correctness and intended behavior. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018. URL https://arxiv.org/abs/1707.01495.

Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Chenxiao Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Qizheng Gu, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yang Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Haoyu Lu, Lijun Lu, Yashuo Luo, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Zeyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Lin Sui, Xinjie Sun, Flood Sung, Yunpeng Tai, Heyi Tang, Jiawen Tao, Qifeng Teng, Chaoran Tian, Chensi Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Si Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Haoning Wu, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Jing Xu, Jing Xu, Junjie Yan, Yuzi Yan, Hao Yang, Xiaofei Yang, Yi Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Siyu Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Shaojie Zheng, Longguang Zhong, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2026. URL https://arxiv.org/abs/2507.20534.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023. URL https://arxiv.org/abs/1706.03741.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena, 2025. URL https://arxiv.org/abs/2504.11442.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, 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, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li,

H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, 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, Mingxu Zhou, 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, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, 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, 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 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Bowen Jiang, Taiwei Shi, Ryo Kamoi, Yuan Yuan, Camillo J. Taylor, Longqi Yang, Pei Zhou, and Sihao Chen. One model, all roles: Multi-turn, multi-agent self-play reinforcement learning for conversational social intelligence, 2026. URL https://arxiv.org/abs/2602.03109.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

David A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. FT Press, Upper Saddle River, NJ, 2 edition, 2014. ISBN 9780133892505.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang

Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

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 Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Taiwei Shi, Kai Chen, and Jieyu Zhao. Safer-instruct: Aligning language models with automated preference data, 2024. URL https://arxiv.org/abs/2311.08685.

Taiwei Shi, Zhuoer Wang, Longqi Yang, Ying-Chun Lin, Zexue He, Mengting Wan, Pei Zhou, Sujay Jauhar, Sihao Chen, Shan Xia, Hongfei Zhang, Jieyu Zhao, Xiaofeng Xu, Xia Song, and Jennifer Neville. Wildfeedback: Aligning llms with in-situ user interactions and feedback, 2025. URL https://arxiv.org/abs/2408.15549.

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. Efficient reinforcement finetuning via adaptive curriculum learning, 2026. URL https://arxiv.org/abs/2504.05520.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

David Silver and Richard S. Sutton. Welcome to the era of experience. 2025. URL https://api.semanticscholar.org/CorpusID:277919528.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, Alex Makelov, Alex Neitz, Alex Wei, Alexandra Barr, Alexandre Kirchmeyer, Alexey Ivanov, Alexi Christakis, Alistair Gillespie, Allison Tam, Ally Bennett, Alvin Wan, Alyssa Huang, Amy McDonald Sandjideh, Amy Yang, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrei Gheorghe, Andres Garcia Garcia, Andrew Braunstein, Andrew Liu, Andrew Schmidt, Andrey Mereskin, Andrey Mishchenko, Andy Applebaum, Andy Rogerson, Ann Rajan, Annie Wei, Anoop Kotha, Anubha Srivastava, Anushree Agrawal, Arun Vijayvergiya, Ashley Tyra, Ashvin Nair, Avi Nayak, Ben Eggers, Bessie Ji, Beth Hoover, Bill Chen, Blair Chen, Boaz Barak, Borys Minaiev, Botao Hao, Bowen Baker, Brad Lightcap, Brandon McKinzie, Brandon Wang, Brendan Quinn, Brian Fioca, Brian Hsu, Brian Yang, Brian Yu, Brian Zhang, Brittany Brenner, Callie Riggins Zetino, Cameron Raymond, Camillo Lugaresi, Carolina Paz, Cary Hudson, Cedric Whitney, Chak Li, Charles Chen, Charlotte Cole, Chelsea Voss, Chen Ding, Chen Shen, Chengdu Huang, Chris Colby, Chris Hallacy, Chris Koch, Chris Lu, Christina Kaplan, Christina Kim, CJ Minott-Henriques, Cliff Frey, Cody Yu, Coley Czarnecki, Colin Reid, Colin Wei, Cory Decareaux, Cristina Scheau, Cyril Zhang, Cyrus Forbes, Da Tang, Dakota Goldberg, Dan Roberts, Dana Palmie, Daniel Kappler, Daniel Levine, Daniel Wright, Dave Leo, David Lin, David Robinson, Declan Grabb, Derek Chen, Derek Lim, Derek Salama, Dibya Bhattacharjee, Dimitris Tsipras, Dinghua Li, Dingli Yu, DJ Strouse, Drew Williams, Dylan Hunn, Ed Bayes, Edwin Arbus, Ekin Akyurek, Elaine Ya Le, Elana Widmann, Eli Yani, Elizabeth Proehl, Enis Sert, Enoch Cheung, Eri Schwartz, Eric Han, Eric Jiang, Eric Mitchell, Eric Sigler, Eric Wallace, Erik Ritter, Erin Kavanaugh, Evan Mays, Evgenii Nikishin, Fangyuan Li, Felipe Petroski Such, Filipe de Avila Belbute Peres, Filippo Raso, Florent Bekerman, Foivos Tsimpourlas, Fotis Chantzis, Francis Song, Francis Zhang, Gaby Raila, Garrett McGrath, Gary Briggs, Gary Yang, Giambattista Parascandolo, Gildas Chabot, Grace Kim, Grace Zhao, Gregory Valiant, Guillaume Leclerc, Hadi Salman, Hanson Wang, Hao Sheng, Haoming Jiang, Haoyu Wang, Haozhun Jin, Harshit Sikchi, Heather Schmidt, Henry Aspegren, Honglin Chen, Huida Qiu, Hunter Lightman, Ian Covert, Ian Kivlichan, Ian Silber, Ian Sohl, Ibrahim Hammoud, Ignasi Clavera, Ikai Lan, Ilge Akkaya, Ilya Kostrikov, Irina Kofman, Isak Etinger, Ishaan Singal, Jackie Hehir, Jacob Huh, Jacqueline Pan, Jake Wilczynski, Jakub Pachocki, James Lee, James Quinn, Jamie Kiros, Janvi Kalra, Jasmyn Samaroo, Jason Wang, Jason Wolfe, Jay Chen, Jay Wang, Jean Harb, Jeffrey Han, Jeffrey Wang, Jennifer Zhao, Jeremy Chen, Jerene Yang, Jerry Tworek, Jesse Chand, Jessica Landon, Jessica Liang, Ji Lin, Jiancheng Liu, Jianfeng Wang, Jie Tang, Jihan Yin, Joanne Jang, Joel Morris, Joey Flynn, Johannes Ferstad, Johannes Heidecke, John Fishbein, John Hallman, Jonah Grant, Jonathan Chien, Jonathan Gordon, Jongsoo Park, Jordan Liss, Jos Kraaijeveld, Joseph Guay, Joseph Mo, Josh Lawson, Josh McGrath, Joshua Vendrow, Joy Jiao, Julian Lee, Julie Steele, Julie Wang, Junhua Mao, Kai Chen, Kai Hayashi, Kai Xiao, Kamyar Salahi, Kan Wu, Karan Sekhri, Karan Sharma, Karan Singhal, Karen Li, Kenny Nguyen, Keren Gu-Lemberg, Kevin King, Kevin Liu, Kevin Stone, Kevin Yu, Kristen Ying, Kristian Georgiev, Kristie Lim, Kushal Tirumala, Kyle Miller, Lama Ahmad, Larry Lv, Laura Clare, Laurance Fauconnet, Lauren Itow, Lauren Yang, Laurentia Romaniuk, Leah Anise, Lee Byron, Leher Pathak, Leon Maksin, Leyan Lo, Leyton Ho, Li Jing, Liang Wu, Liang Xiong, Lien Mamitsuka, Lin Yang, Lindsay McCallum, Lindsey Held, Liz Bourgeois, Logan Engstrom, Lorenz Kuhn, Louis Feuvrier, Lu Zhang, Lucas Switzer, Lukas Kondraciuk, Lukasz Kaiser, Manas Joglekar, Mandeep Singh, Mandip Shah, Manuka Stratta, Marcus Williams, Mark

Chen, Mark Sun, Marselus Cayton, Martin Li, Marvin Zhang, Marwan Aljubeh, Matt Nichols, Matthew Haines, Max Schwarzer, Mayank Gupta, Meghan Shah, Melody Huang, Meng Dong, Mengqing Wang, Mia Glaese, Micah Carroll, Michael Lampe, Michael Malek, Michael Sharman, Michael Zhang, Michele Wang, Michelle Pokrass, Mihai Florian, Mikhail Pavlov, Miles Wang, Ming Chen, Mingxuan Wang, Minnia Feng, Mo Bavarian, Molly Lin, Moose Abdool, Mostafa Rohaninejad, Nacho Soto, Natalie Staudacher, Natan LaFontaine, Nathan Marwell, Nelson Liu, Nick Preston, Nick Turley, Nicklas Ansman, Nicole Blades, Nikil Pancha, Nikita Mikhaylin, Niko Felix, Nikunj Handa, Nishant Rai, Nitish Keskar, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Oona Gleeson, Pamela Mishkin, Patryk Lesiewicz, Paul Baltescu, Pavel Belov, Peter Zhokhov, Philip Pronin, Phillip Guo, Phoebe Thacker, Qi Liu, Qiming Yuan, Qinghua Liu, Rachel Dias, Rachel Puckett, Rahul Arora, Ravi Teja Mullapudi, Raz Gaon, Reah Miyara, Rennie Song, Rishabh Aggarwal, RJ Marsan, Robel Yemiru, Robert Xiong, Rohan Kshirsagar, Rohan Nuttall, Roman Tsiupa, Ronen Eldan, Rose Wang, Roshan James, Roy Ziv, Rui Shu, Ruslan Nigmatullin, Saachi Jain, Saam Talaie, Sam Altman, Sam Arnesen, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Sarah Yoo, Savannah Heon, Scott Ethersmith, Sean Grove, Sean Taylor, Sebastien Bubeck, Sever Banesiu, Shaokyi Amdo, Shengjia Zhao, Sherwin Wu, Shibani Santurkar, Shiyu Zhao, Shraman Ray Chaudhuri, Shreyas Krishnaswamy, Shuaiqi, Xia, Shuyang Cheng, Shyamal Anadkat, Simón Posada Fishman, Simon Tobin, Siyuan Fu, Somay Jain, Song Mei, Sonya Egoian, Spencer Kim, Spug Golden, SQ Mah, Steph Lin, Stephen Imm, Steve Sharpe, Steve Yadlowsky, Sulman Choudhry, Sungwon Eum, Suvansh Sanjeev, Tabarak Khan, Tal Stramer, Tao Wang, Tao Xin, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Degry, Thomas Shadwell, Tianfu Fu, Tianshi Gao, Timur Garipov, Tina Sriskandarajah, Toki Sherbakov, Tomer Kaftan, Tomo Hiratsuka, Tongzhou Wang, Tony Song, Tony Zhao, Troy Peterson, Val Kharitonov, Victoria Chernova, Vineet Kosaraju, Vishal Kuo, Vitchyr Pong, Vivek Verma, Vlad Petrov, Wanning Jiang, Weixing Zhang, Wenda Zhou, Wenlei Xie, Wenting Zhan, Wes McCabe, Will DePue, Will Ellsworth, Wulfie Bain, Wyatt Thompson, Xiangning Chen, Xiangyu Qi, Xin Xiang, Xinwei Shi, Yann Dubois, Yaodong Yu, Yara Khakbaz, Yifan Wu, Yilei Qian, Yin Tat Lee, Yinbo Chen, Yizhen Zhang, Yizhong Xiong, Yonglong Tian, Young Cha, Yu Bai, Yu Yang, Yuan Yuan, Yuanzhi Li, Yufeng Zhang, Yuguang Yang, Yujia Jin, Yun Jiang, Yunyun Wang, Yushi Wang, Yutian Liu, Zach Stubenvoll, Zehao Dou, Zheng Wu, and Zhigang Wang. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, Ran Xu, and Caiming Xiong. Coact-1: Computer-using agents with coding as actions, 2025a. URL https://arxiv.org/abs/2508.03923.

Linxin Song, Taiwei Shi, and Jieyu Zhao. The hallucination tax of reinforcement finetuning, 2025b. URL https://arxiv.org/abs/2505.13988.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J. Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback, 2026. URL https://arxiv.org/abs/2602.02482.

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-21b81902c146819db63cd98a54ba5f3 2025. Notion Blog.

Sai Wang, Yu Wu, and Zhongwen Xu. Cogito, ergo ludo: An agent that learns to play by reasoning and planning, 2025. URL https://arxiv.org/abs/2509.25052.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger

Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models, 2026. URL https://arxiv.org/abs/2602.12275.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience, 2025. URL https://arxiv.org/abs/2510.08558.

A Full Algorithm and Gated Reflection

Gated Reflection. Algorithm 2 presents the full version of ERL used in our experiments. Compared to the simplified version in Algorithm 1, the key difference is a gating mechanism on the second attempt: reflection and refinement are triggered only when the first-attempt reward satisfies $r^{(1)} < \tau$, where $\tau = 1$. In other words, reflection is applied only to failed or suboptimal trajectories. In early experiments, we applied reflection to all trajectories, including successful ones, but this led to unstable training and reduced generalization. First, reflecting on already successful attempts encouraged reward hacking: the model sometimes generated instance-specific shortcuts that guaranteed success for the current sample but did not generalize to future episodes. Second, early in training when first-attempt rewards are typically low, the optimization signal became dominated by the second attempt and reflection, which are inherently off-policy relative to the base policy. This imbalance weakened the on-policy learning signal and destabilized the policy. The gating mechanism mitigates these issues by ensuring that successful trajectories remain purely on-policy, while reflection is reserved for corrective revision on failed attempts. This design also aligns training with deployment: at inference time, the model must generate $y \sim \pi_\theta(\cdot \mid x)$ without access to reflection Δ or feedback signals. By restricting reflection to corrective cases and preserving sufficient on-policy updates in every batch, the gating mechanism improves stability in training.

Memory Extensions. Algorithm 2 also maintains a simple reflection memory that stores successful reflections as system prompt in plain text. A natural extension is to replace this mechanism with a more sophisticated agentic memory system. For example, before the reflection step (Alg. 2, Line 12), the model may retrieve relevant past reflections from a memory base conditioned on the current input x, and after a successful refinement, update the memory using a structured agentic memory update rule rather than direct overwrite. Such retrieval-and-update schemes would allow ERL to scale to more diverse and long-horizon tasks by enabling selective reuse and continual refinement of past corrective knowledge.

On-Policy Distillation. The internalization step in Algorithm 2 can also be generalized beyond supervised distillation. Instead of training π_θ to reproduce $y^{(2)}$ from x using a standard distillation loss, one may adopt an on-policy reverse KL objective. Let the contextual policy with access to reflection and memory be $\pi_\theta(\cdot \mid x, \Delta)$, and the deployment policy be $\pi_\theta(\cdot \mid x)$. An on-policy distillation objective can be written as

$$\mathcal{L}_{\mathrm{OD}}(\theta) := \mathbb{E}_{x\sim\mathcal{D}}\Big[\mathbb{I}\Big(r^{(2)} > \tau\Big)\ \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[\mathrm{KL}(\pi_\theta(\cdot \mid x, \Delta) \,\|\, \pi_\theta(\cdot \mid x))\big]\Big].$$

which encourages the deployment policy to match the richer contextual policy while remaining on-policy with respect to $\pi_\theta(\cdot \mid x)$. This connects ERL to recent reverse-KL and on-policy distillation approaches (Agarwal et al., 2024; Hübotter et al., 2026; Song et al., 2026; Ye et al., 2026) and provides a principled alternative to supervised internalization.

B Qualitative Analysis of Reflection and Behavioral Revision

To better understand how Experiential Reinforcement Learning (ERL) improves policy learning beyond scalar reward optimization, we conduct a qualitative analysis of the learned reflections across environments. We analyze representative examples from FrozenLake (Tables 2 and 3), Sokoban (Tables 4 and 5), and HotpotQA (Tables 6 and 7). Across these settings, we find that reflections consistently evolve into structured, task-specific abstractions that (1) infer latent environment rules, (2) encode reusable strategies, and (3) translate feedback into actionable behavioral constraints. These properties help explain the substantial performance gains observed in Table 1 and the learning dynamics in Figures 4 and 6.

From feedback to latent rule induction. In sparse-reward environments such as FrozenLake, reflections frequently reconstruct implicit environment dynamics from experience. As shown in Table 2, the model infers symbolic semantics (e.g., identifying `C` as terminal hazards and `B` as the goal) despite receiving no explicit rules during training. This demonstrates that ERL enables the model to transform outcome feedback into an explicit internal representation of environment structure.

Algorithm 2 Reinforcement Learning from Self-Reflection (Full Version)

Inputs: Language model π_θ; dataset of questions x; environment returning feedback f and reward r, reward threshold τ.
Initialize: reflection memory $m \leftarrow \emptyset$.
repeat
  Sample question x from dataset.
  // First attempt
  Sample answer $y^{(1)} \sim \pi_\theta(\cdot \mid x)$.
  Obtain feedback and reward $(f^{(1)}, r^{(1)})$.
  // RL update on the first attempt
  Update θ via $\mathcal{L}_{\text{policy}}(\theta)$ over the first attempt
  // Gated second attempt
  if $r^{(1)} < \tau$ **then**
    // Reflection with cross-episode memory
    Sample reflection $\Delta \sim \pi_\theta(\cdot \mid x, y^{(1)}, f^{(1)}, r^{(1)}, m)$.
    Sample refined answer $y^{(2)} \sim \pi_\theta(\cdot \mid x, \Delta)$.
    Obtain feedback and reward $(f^{(2)}, r^{(2)})$.
    Set reflection reward $\tilde{r} \leftarrow r^{(2)}$.
    // Store reflection only if improved beyond threshold
    if $r^{(2)} > \tau$ **then**
      Store reflection: $m \leftarrow \Delta$.
    end if
    // RL update on the second attempt
    Update θ via $\mathcal{L}_{\text{policy}}(\theta)$ over reflection and second attempt.
    // Internalization
    Update θ via $\mathcal{L}_{\text{distill}}(\theta)$ to internalize reflection, training π_θ to produce $y^{(2)}$ from x only.
  end if
until converged

Crucially, this process does not merely restate observations, but organizes them into a coherent rule system (e.g., transition constraints, reward conditions, and termination criteria). Compared to RLVR, where such structure must be implicitly encoded through repeated trial-and-error, ERL externalizes this inference into a reusable intermediate representation, reducing the burden on implicit credit assignment. This interpretation is supported by the ablation results in Table 1: removing structured reflection (ERL w/o Refl.) leads to substantial performance degradation across tasks, particularly in environments requiring latent rule discovery such as FrozenLake and Sokoban. Without reflection, the model loses the ability to explicitly construct and reuse these rule-level abstractions, reverting to inefficient trial-and-error optimization similar to RLVR. This suggests that the primary benefit of ERL arises not only from additional optimization signal, but from the explicit representation of inferred environment structure that guides subsequent behavior.

Emergence of reusable strategy primitives. Beyond rule induction, reflections encode high-level strategies that generalize across states. Table 3 shows that the model develops reusable heuristics such as avoiding hazards, prioritizing goal-directed movement, and systematically exploring safe actions. Similarly, in Sokoban (Tables 4–5), reflections capture domain-specific planning principles such as goal-directed box pushing and avoiding irreversible states.

These strategies exhibit two important properties. First, they are abstract: they apply across different grid configurations rather than overfitting to a single trajectory. Second, they are operational: they directly constrain future actions (e.g., "never push a box into a wall"). This suggests that ERL effectively converts episodic experience into policy-level inductive biases, which can then be internalized through distillation.

Reflection as structured credit assignment. A key qualitative difference between ERL and RLVR lies in how failures are interpreted. In RLVR, failure signals remain scalar and must be propagated backward through many steps. In contrast, ERL reflections explicitly attribute failure to specific causes and propose targeted corrections. For example, reflections frequently identify invalid moves (e.g., out-of-bounds actions or hazardous transitions) and prescribe precise alternatives.

This aligns with the quantitative evidence in Figure 6, where post-reflection trajectories consistently outperform pre-reflection ones. Qualitatively, this improvement arises because reflections decompose global reward signals into localized behavioral adjustments, effectively serving as a structured credit-assignment mechanism within each episode.

Task-dependent structure of reflections. The form of reflection adapts to the nature of the task. In control environments like FrozenLake and Sokoban, reflections emphasize spatial reasoning, constraints, and planning heuristics. In contrast, HotpotQA reflections (Tables 6–7) focus on epistemic reasoning, including retrieval verification, evidence grounding, and hallucination avoidance.

Notably, HotpotQA reflections encode strict procedural rules (e.g., mandatory retrieval, evidence verification, and explicit uncertainty handling), which function as a form of self-imposed reasoning discipline. This difference in structure helps explain why ERL yields smaller but consistent gains in HotpotQA: the task already provides denser supervision, and reflections primarily refine reasoning reliability rather than discovering latent dynamics.

C Envrionment Configuration Details

C.1 Frozen Lake

Frozen Lake is a grid-based navigation environment in which an agent must move from a start location to a goal location on an $n \times n$ grid. We configure our FrozenLake environment following a setup similar to those used in TextArena (Guertler et al., 2025) and Wang et al. (2025). The grid size n is sampled uniformly from $[2, 9]$. For each instance, the start and goal tiles are randomly selected as distinct positions. The grid layout is generated procedurally to ensure that at least one valid path exists between the start and goal.

Each non-goal tile is assigned as either a safe frozen tile or a hole according to a frozen-tile probability parameter p, sampled uniformly from $[0.6, 0.85)$. Holes represent terminal failure states, while frozen tiles are traversable. Transitions are deterministic: the agent's chosen action directly determines its next grid position, subject to boundary constraints.

At every step, the agent observes a full textual representation of the grid. To reduce the influence of pretrained symbolic priors, we employ abstract symbols rather than semantically meaningful markers. The default encoding is:

$$\texttt{A} = \text{agent position}, \quad \texttt{B} = \text{goal tile}, \quad \texttt{C} = \text{hole}, \quad \texttt{D} = \text{safe frozen tile}.$$

This representation encourages the model to infer environment dynamics through interaction rather than relying on prior associations.

In addition to the textual presentation of the grid, the environment also appends structured textual feedback to the end of the interaction history after each action. This feedback communicates the outcome of the most recent transition and serves as the only explicit signal describing terminal or invalid events. The feedback messages are defined as follows:

- `The agent reached the goal` — issued when the agent successfully enters the goal tile. The episode terminates with reward 1.0.
- `The agent fell into the hole` — issued when the agent enters a hole tile. The episode terminates with reward 0.0.
- `Hit the max step limit` — issued when the agent exhausts the fixed step budget. The episode terminates with reward 0.0.
- `No valid actions were recorded.` — issued when the agent produces an invalid action or when the attempted action results in no state change, such as moving into a boundary. The episode continues unless the step budget is exhausted.

The default system prompt, self-reflection prompt, and example task are shown in Tables 8, 10, and 12.

The reward function is sparse. The agent receives a reward of 1.0 if it reaches the goal tile and 0.0 otherwise. Episodes terminate upon reaching the goal, entering a hole, or exhausting a fixed step budget of 8 actions.

FrozenLake Self-reflection Example (Part 1)

<**game_rules**>
1. Symbol Meanings:

- A: Player/agent starting position (initial location)
- B: Goal or safe terminal state (reaching this cell ends the game successfully)
- C: Obstacle or wall (cannot pass through; movement into this cell results in failure)
- D: Empty or neutral terrain (safe to move to, no reward or penalty)

2. Information & Interpretation:

- The grid is nxn. The agent can move in four directions: Up, Down, Left, Right.
- Movement into a cell marked 'C' results in immediate termination with a reward of 0.00 and failure.
- The agent cannot move outside the grid (boundaries are walls).
- The agent starts at the position marked 'A'.
- 'B' is the only terminal goal — reaching it results in a successful episode and a positive reward (implied by successful trajectories).
- 'C' and 'B' are static and unchanging throughout the game.

3. Gameplay & Actions:

- The agent selects one of four directions: Up, Down, Left, Right.
- If the next cell is within bounds and not marked 'C', the agent moves there.
- If the next cell is 'C' or out of bounds, the move is blocked and has no effect.
- The agent receives no reward for movement unless it reaches 'B'.

4. Action Effects:

- Moving into a 'C' cell: game ends immediately with reward = 0.00 and success = False.
- Moving into a valid adjacent cell (not 'C' and within bounds): agent moves to that cell.
- Moving out of bounds: no effect (move is invalid).
- Reaching a cell marked 'B': game ends with reward = 1.00 and success = True.

5. Game Objective & Termination:

- The objective is to reach the cell marked 'B' without stepping on any 'C'.
- The game terminates when the agent reaches 'B' (success) or steps on 'C' (failure).
- Rewards:
  - +1.00 for reaching 'B'
  - 0.00 for stepping on 'C' or ending due to out-of-bounds movement
  - 0.00 for all other movements
- Episode ends when either success or failure occurs.

</**game_rules**>

Table 2: FrozenLake self-reflection example, taken from step 40 of Qwen3-4B-Instruct-2507 training.

For training, we generate 10,000 procedurally sampled instances. Evaluation is conducted on a disjoint set of 100 instances constructed using the same generation process.

FrozenLake Self-reflection Example (Part 2)

<**Strategy**>:
1. Core Strategies:

- Always avoid moving into any cell marked 'C' — stepping on it results in immediate failure.
- Never move in a direction that would take the agent out of bounds.
- Identify the agent's current position and assess all adjacent cells to determine safe moves.
- Prioritize movement toward the goal 'B' if it is adjacent or reachable through a clear path.
- If multiple safe moves exist, choose one that brings the agent closer to 'B' or opens a new path toward it.
- If no path to 'B' exists, the agent should remain in a safe zone, avoiding walls and boundaries.

2. Tactical Tips:

- Always check the current grid state before acting — verify adjacent cells for walls or out-of-bounds.
- If the agent is on the edge of the grid, do not move in the direction that would go out of bounds.
- If 'B' is adjacent, move directly toward it.
- Avoid repeated movement in the same direction — this leads to stagnation or looping.
- Explore cautiously: if multiple safe directions exist, use a systematic pattern (e.g., clockwise) to avoid missing a path to 'B'.
- If the agent is stuck in a loop (e.g., between two walls), it has failed to reach 'B' and should be re-evaluated.

</**Strategy**>

Always remember:

- Valid actions: Up, Down, Left, Right.
- Think step by step, but make the final line only the next action wrapped in triple backticks, e.g., ```` ```Up``` ````.

Table 3: FrozenLake self-reflection example (continued), taken from step 40 of Qwen3-4B-Instruct-2507 training.

C.2 SOKOBAN

Sokoban is a grid-based box-pushing environment in which an agent must place all boxes onto designated goal tiles. We configure our Sokoban environment following a setup similar to those used in TextArena (Guertler et al., 2025) and Wang et al. (2025). Each instance is represented as an $n \times n$ grid, where n is sampled uniformly from $[6, 8]$ in our procedural generator. We construct single-box, single-goal layouts with border walls, and randomly sample interior positions for the goal, box, and player, subject to non-overlap constraints.

To control difficulty, each generated layout is accepted only if its shortest valid solution is at most 8 moves (computed by BFS over player–box states). This guarantees solvability while keeping episodes short-horizon. Train and test splits are disjoint at the layout level.

Sokoban Self-reflection Example (Part 1)

<**game_rules**>

1. Symbol Meanings:

- `A` = Player agent
- `B` = Box (movable object)
- `C` = Goal tile (target area for boxes)
- `D` = Empty floor tile
- `E` = Wall (impassable)

2. Information & Interpretation:

- The presence of `B` indicates movable boxes; their positions are critical to assess.
- The presence of `C` defines the goal locations.
- The agent (`A`) is the only entity that can move or push boxes.
- All boxes must be pushed onto `C` tiles to win.

3. Gameplay & Actions:

- The agent can move in four directions: Up, Down, Left, Right.
- When the agent moves into a space adjacent to a box (`B`), the box is pushed one tile in that direction.
- The agent can only push a box if the space beyond the box is empty (`D`) or a goal (`C`).
- If the space beyond the box is blocked (wall or another box), the push fails and the agent does not move.

4. Action Effects:

- Moving into an empty tile (`D`): agent moves one tile in direction.
- Moving into a box (`B`) and pushing it into an empty tile (`D`) or goal (`C`): box moves; agent moves.
- Attempting to push a box into a wall (`E`) or another box: push fails, agent does not move.

5. Game Objective & Termination:

- Win condition: All boxes are on goal tiles (`C`).
- Loss condition: Agent reaches max steps without progress or makes no progress in pushing boxes.
- Game ends when all boxes are on `C` tiles, or after max_steps are reached.

</**game_rules**>

Table 4: Sokoban self-reflection example, taken from step 90 of Qwen3-4B-Instruct-2507 training.

At every step, the agent observes the full textual grid. As in FrozenLake, we use abstract symbols to reduce direct reliance on pretrained semantic priors. The default encoding is:

$$\texttt{A} = \text{agent position}, \quad \texttt{a} = \text{agent on box}, \quad \texttt{B} = \text{box}, \quad \texttt{b} = \text{box on goal},$$
$$\texttt{C} = \text{goal tile}, \quad \texttt{E} = \text{wall}, \quad \texttt{D} = \text{floor}.$$

The action space is $\{\texttt{Up}, \texttt{Down}, \texttt{Left}, \texttt{Right}\}$. Moves are deterministic. The agent may push exactly one adjacent box only when the cell behind the box is free; it cannot pull boxes, move through walls, or move through boxes. Invalid moves produce no state change.

Sokoban Self-reflection Example (Part 2)

<**strategy**>
1. Core Strategies:

- **Prioritize box placement on goal tiles (`C`)**: Only attempt to move boxes that are directly adjacent to a `C`. This is the highest-value action.
- **Avoid unnecessary movement**: Do not move the agent unless it is directly adjacent to a box or a goal.
- **Minimize risk of failure**: Never attempt to push a box into a wall or another box unless absolutely necessary.
- **Map the environment**: Identify all boxes and goals early. Only act when a box is adjacent to a `C`.

2. Tactical Tips:

- If a box is adjacent to a `C`, immediately push it into the `C` — this is the highest-value action.
- If a box is not adjacent to a `C`, assess whether moving the agent closer to it will allow a direct push.
- If no box is adjacent to a goal, do not move — waiting may allow a better opportunity to push.
- Never attempt to push a box into a wall or another box — it wastes steps and increases failure risk.

Always remember:

- Valid actions: Up, Down, Left, Right.
- Only push a box if the space behind it is free.
- Think step by step, but make the final line only the next action wrapped in triple backticks, e.g., ```` ```Up``` ````.

</**strategy**>

Table 5: Sokoban self-reflection example (continued), taken from step 90 of Qwen3-4B-Instruct-2507 training.

In addition to the grid observation, the interaction trace includes structured textual transition feedback after each action. The feedback messages are:

- `The agent solved the puzzle (all boxes on goals).` — issued when all boxes are on goal tiles. The episode terminates with reward 1.0.
- `The agent moved or pushed a box; puzzle not solved yet.` — issued when the action changes the state but the puzzle remains unsolved.
- `The agent did not move (likely hit a wall or tried to push into a blocked space).` — issued when the chosen move is ineffective (no state change).
- `Hit the max step limit` — issued when the fixed step budget is exhausted before solving. The episode terminates with reward 0.0.

The default system prompt, self-reflection prompt, and example task are shown in Tables 8, 10, and 13.

The reward is sparse: 1.0 if and only if all boxes are on goals, and 0.0 otherwise. Episodes terminate on success or when the step budget is exhausted. In the generated REEX Sokoban dataset, the per-instance step budget is 8.

HotpotQA Self-reflection Example (Part 1)

You are a helpful assistant who answers questions accurately by retrieving relevant information when the answer is not directly stated. Always perform a search if the question requires external knowledge or involves specific details not explicitly given.

If the question involves identifying a film, person, event, organization, or year based on criteria (e.g., director, cast, genre, location, role), first verify whether such an entity exists and matches all stated conditions using a targeted search.

After retrieving results, analyze them step by step to confirm:

- Whether the entity (e.g., film, person, event, organization) matches the specified criteria.
- Whether any of the documents contain conflicting or irrelevant information.
- Whether the required detail (e.g., year, name, location, role) is explicitly stated.

If no match is found or the required detail is missing, clearly state that the information is not available or that no such entity exists based on the evidence. Never hallucinate facts.

Always ensure that the final answer is directly supported by retrieved documents. If a value is not explicitly mentioned in any document, do not assume or infer it — instead, state that it is unknown.

If the question contains inaccuracies or misleading assumptions, explicitly identify and correct them. Do not accept false premises.

Critical Rules:

- Never skip retrieval when the answer is not directly stated. Always search for external facts when the question references specific people, roles, events, or relationships.
- Always verify the identity and role of a person using precise, context-specific queries (e.g., include the organization, event, or title). Do not rely on indirect references or partial matches.
- Do not assume relationships (e.g., professional roles, affiliations, or personal connections) unless explicitly confirmed by retrieved evidence.
- If a name may refer to multiple entity types (e.g., person, publication, organization), verify its identity before proceeding. Do not misclassify entities.
- Always verify key attributes (e.g., genre, location, role, specifications, rankings, population) using explicit searches. Do not infer or generalize from context.
- When comparing multiple entities, verify each one independently. Do not assume shared attributes without confirmation.
- If a document uses vague or approximate language (e.g., "nearly," "about"), do not convert it into a precise value. Only use explicitly stated facts.
- If required information is not explicitly supported by evidence, the answer must be `\boxed{Unknown}`.

Table 6: HotpotQA self-reflection example, taken from step 70 of Qwen3-4B-Instruct-2507 training.

For training, we generate 10,000 procedurally sampled instances. Evaluation is conducted on a disjoint set of 100 instances built with the same generation process.

HotpotQA Self-reflection Example (Part 2)

Final Answer Format:
Provide your final answer in \boxed{} format. If a specific fact is supported by evidence, provide it clearly and correct any inaccuracies in the question if necessary.

Available tool

```
[
  {
    "type": "function",
    "function": {
      "name": "local_search",
      "description": "Search for information using a dense
                      retrieval server with Wikipedia corpus",
      "parameters": {
        "type": "object",
        "properties": {
          "query": {
            "type": "string",
            "description": "Search query to retrieve relevant
                            documents"
          },
          "top_k": {
            "type": "integer",
            "description": "Number of results to return (default: 5)",
            "minimum": 1,
            "maximum": 50
          }
        },
        "required": ["query"]
      }
    }
  }
]
```

Table 7: HotpotQA self-reflection example (continued), taken from step 70 of Qwen3-4B-Instruct-2507 training.

C.3 HOTPOTQA

HotpotQA is a multi-hop open-domain question answering task in which an agent must answer compositional questions by retrieving and synthesizing evidence across multiple documents. Each instance consists of a natural-language question and a reference answer.

Unlike grid-based control environments such as FrozenLake or Sokoban, HotpotQA does not expose an explicit environment state. Instead, the agent operates through a tool-augmented interaction loop in which it alternates between reasoning, retrieval, and answer generation. The agent may invoke an external retrieval tool and ultimately produce a final textual answer. The solver instruction requires that the final answer be formatted inside \boxed{} to enable reliable extraction.

The retrieval interface is defined as:

`local_search(query, top_k)`,

which queries a local dense-retrieval server built over an indexed Wikipedia corpus and returns ranked text snippets relevant to the query. We use a Wikipedia corpus organized by `PeterJinGo/wiki-18-corpus`, with prebuilt dense indices from `PeterJinGo/wiki-18-e5-index`. Embeddings are generated using

Initial System Prompt for FrozenLake and Sokoban

You are an agent playing a game on a grid, acting as a reasoning engine.
Your decisions are based on your current game rules (your best guess of how the game works) and your strategic playbook (your learned strategies). These may be incomplete or incorrect.
Your only way to interact with the environment is by choosing your NEXT ACTION.

Instructions
1. Analyze State: Summarize the current state.
2. Predict Long-term Value of Outcomes: Evaluate the strategic value and potential of the current state for the future.
3. Predict Immediate Consequences: For the top two candidate actions, predict their consequences using a "result-because" structure.
4. Select the Best Action: Choose the action leading to the most advantageous future state.

Required response structure
```
<reason>
**1. Analysis of the Current State:** [Summary of the board state.]
**2. Prediction of the Value of Current States:** [Assessment] - Value: High / Medium / Low
**3. Prediction of Immediate Consequences:** [Top 2 candidate actions]
</reason>
```

Then output the NEXT ACTION inside triple backticks, e.g., ```` ```Up``` ````.

Always remember:
- Valid actions: Up, Down, Left, Right.
- Think step by step, but make the final line only the next action wrapped in triple backticks.

Table 8: Initial System used for FrozenLake and Sokoban.

`intfloat/e5-base-v2`. Retrieval is powered by FAISS (Douze et al., 2024) with multi-GPU support. During each episode, the agent is allowed up to 5 interaction turns, which may include reasoning steps, tool calls, and final answer submission.

Following the evaluation protocol of Search-R1 (Jin et al., 2025), the answer extracted from `\boxed{}` is normalized prior to scoring by lowercasing and whitespace canonicalization. Correctness is measured using token-level F1 against the ground-truth answer. The reward function assigns a score of 1.0 for exact matches, a proportional reward equal to the F1 score when the F1 is at least 0.3, and 0 otherwise.

The default system prompt, self-reflection prompt, and example task are shown in Tables 9, 11, and 14.

D Training Configuration Details

We train all models with the rLLM agent training stack (Tan et al., 2025) using GRPO (Shao et al., 2024). Training runs on a single node with 8 H100s and uses vLLM (Kwon et al., 2023) with FlashAttention (Dao, 2024).

We enable hybrid engine training, gradient checkpointing, and remove-padding. The optimizer learning rate is `1e-6`. Actor updates use a mini batch size of 64, dynamic batch sizing, and a max token length per GPU of 24,000. FSDP parameter/optimizer offload is enabled for the actor, and parameter offload is enabled for the reference model.

Initial System Prompt for HotpotQA

You are a helpful assistant who answers questions directly and efficiently.

Provide your final answer in \boxed{} format.

Available tool

```
[
  {
    "type": "function",
    "function": {
      "name": "local_search",
      "description": "Search for information using a dense
                      retrieval server with Wikipedia corpus",
      "parameters": {
        "type": "object",
        "properties": {
          "query": {
            "type": "string",
            "description": "Search query to retrieve relevant
                            documents"
          },
          "top_k": {
            "type": "integer",
            "description": "Number of results to return
                            (default: 5)",
            "minimum": 1,
            "maximum": 50
          }
        },
        "required": ["query"]
      }
    }
  }
]
```

Table 9: Initial system prompt used for HotpotQA.

We set the training batch size to 64, with a maximum prompt length of 8,196 tokens and a maximum response length of 8,196 tokens. Rollouts are generated asynchronously using vLLM in async mode with a tensor model parallel size of 1. We use a sampling temperature of 0.7, GPU memory utilization of 0.85. For validation rollouts, we generate 4 samples per prompt with temperature 0.7, top-p sampling set to 0.8, and top-k sampling set to 20.

KL regularization is enabled using a low-variance KL loss with coefficient 0.001, and we use a fixed KL control coefficient of 0.001. The actor clipping ratio upper bound is set to 0.28, and the entropy coefficient is set to 0. Rejection sampling and stepwise advantage estimation are disabled.

For RLVR training, we generate 10 samples per prompt. For ERL training, we generate only 4 samples per prompt for each attempt to match the compute budget of RLVR. Evaluation is performed every 5 iterations, and training is manually early stopped upon convergence.

The design and implementation details of the ERL algorithm can be found in Appendix A.

Self-reflection Prompt for Frozen Lake and Sokoban

You are a chief scientific strategist and master tactician.
Your mission is to analyze extensive field data from numerous operations to distill and refine the Master Rulebook of a complex game.
You will be presented with a large collection of highly successful trajectories and critical failure trajectories, collected over a long period.
Your primary task is to perform a deep, comparative analysis to understand the fundamental principles of victory and defeat.
Act as a grand strategist, identifying universal patterns and high-level causal relationships.
Your goal is to synthesize these insights to produce the next generation's Master Rulebook, making it more robust, accurate, and effective.

Core Principles:
- Think Long-Term: focus on universal, strategic truths that hold across diverse scenarios.
- Learn from Contrast: extract insights by comparing winners and losers.
- Synthesize and Consolidate: produce a single unified theory.
- Be Authoritative and Concise: state rules as definitive principles.

Your output MUST be a single consolidated <prompt> block representing the new Master Rulebook:

```
<prompt>
<game_rules>
**1. Symbol Meanings:** [...]
**2. Information & Interpretation:** [...]
**3. Gameplay & Actions:** [...]
**4. Action Effects:** [...]
**5. Game Objective & Termination:** [...]
</game_rules>
<strategy>
**1. Core Strategies:** [...]
**2. Tactical Tips:** [...]
</strategy>
</prompt>
```

Table 10: Self-reflection prompt used for Frozen Lake and Sokoban.

Self-reflection Prompt for HotpotQA

You are an expert prompt updater.
You will analyze recent trajectories, tool calls, and rewards to improve the solver's system prompt.
When failures occur, explicitly add rules that prevent repeating them (e.g., missing tool calls, hallucinated facts, or unboxed final answers).
Keep the prompt short, actionable, and reusable.
Output ONLY the improved system prompt wrapped in <prompt>...</prompt> tags.

Table 11: Self-reflection prompt used for HotpotQA.

Example Task for Frozen Lake

{System Prompt}

```
Current Observation (0):
D   D   C   D
A   D   D   C
D   C   D   D
D   D   B   D
```

You have not achieved the goal yet. Please give the next action.

Action space
`Up | Down | Left | Right`

Output requirement
Return reasoning in `<reason>...</reason>` and final action in triple backticks, e.g., ```` ```Right``` ````.

Table 12: Example Frozen Lake task instance.

Example Task for Sokoban

{System Prompt}

```
Current Board (0):
E   E   E   E   E   E
E   A   D   B   C   E
E   D   D   D   D   E
E   E   E   E   E   E
```

Puzzle not solved yet. Provide the next move.

Action space
`Up | Down | Left | Right`

Output requirement
Return reasoning in `<reason>...</reason>` and final action in triple backticks, e.g., ```` ```Right``` ````.

Table 13: Example Sokoban task instance.

Example Task for HotpotQA

{System Prompt}

Question:
Which university did the author of "The Hobbit" attend?

Table 14: Example HotpotQA task instance.

Second-Attempt Prompt Template for the No-Reflection Variant

{System Prompt}

You are also provided with the model's past attempt data, including observations, actions, rewards, and feedback.
Use this information as context to make a better next-attempt decision policy.
Follow the action/output format exactly.

{First Attempt's Trajectory}

Table 15: Generic second-attempt system prompt used in the no-reflection ablation. The model is provided with the full first-attempt trajectory (observations, actions, rewards, and feedback) together with a generic instruction encouraging improvement, without any structured reflection signal.