BI-LEVEL ADAPTIVE REGULATION OF INTRA- AND INTER-TASK LEARNING FOR COMPACT LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing agents that autonomously complete long-horizon tasks is an important goal in AI research. Repeated calls to large-scale language models during long-horizon interactions can incur high inference costs, motivating the development of capable agents through targeted training of compact language models. However, existing training paradigms provide limited support for adaptive regulation within and across tasks. At the intra-task level, outcome rewards or undifferentiated process rewards offer limited guidance for switching among reasoning modes. At the inter-task level, fixed task distributions can become misaligned with agents' evolving capabilities. As a result, compact agents may persist with unsuitable reasoning modes and repeat ineffective actions, increasing interaction costs without effectively advancing the task. To this end, this paper proposes ARETE (Adaptive REgulation with Tailored Experiences), a long-horizon reinforcement learning framework that combines intra-task reasoning regulation with inter-task practice regulation. For intra-task regulation, inspired by the self-regulated learning (SRL) framework from educational psychology, ARETE organizes reasoning into three modes: forethought, performance, and self-reflection, using differentiated, verifiable feedback to regulate their use. For inter-task regulation, inspired by the zone of proximal development from developmental and educational psychology, ARETE generates verified scaffolded tasks and compares unaided and scaffolded performance to estimate learning potential, guiding capability-matched task generation and practice allocation. Experiments on two complex interactive benchmarks show that ARETE improves compact agents, enabling a 2B-parameter model to perform competitively with models exceeding 100B parameters.

1 INTRODUCTION

Rapid advances in large language models (LLMs) enable agents to tackle complex tasks through sustained interaction with external environments Yao et al. (2023); Xi et al. (2025). Unlike single-step reasoning, long-horizon tasks require agents to coordinate planning, execution, and error correction as observations and action outcomes accumulate Liu et al. (2025). Long-horizon competence therefore depends not only on individual reasoning abilities, but also on regulating their use in response to environmental feedback Bengio et al. (2009); Tzannetos et al. (2023).

However, repeated calls to large-scale models over long interaction sequences incur high inference costs, motivating the development of capable agents built on compact LMs. Reinforcement learning provides a promising route to this goal. AgentGym-RL progressively extends interaction horizons to encourage exploration and stabilize training, while RLVMR introduces verifiable process feedback for planning, exploration, and reflection Xi et al. (2026); Zhang et al. (2026). However, a reasoning mode can be executed correctly even when invoking it is unnecessary or ill-suited to the current task state. Intra-task feedback must therefore distinguish mode applicability from execution quality. Across tasks, fixed distributions may waste training effort on tasks that are already mastered or are currently intractable. Existing curriculum and environment-generation methods adapt training tasks based on agent performance or difficulty Lai et al. (2024); Hu et al. (2025); Zala et al. (2024); Qi et al. (2025); Dong et al. (2026), but a low success rate alone does not reveal whether a task

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.

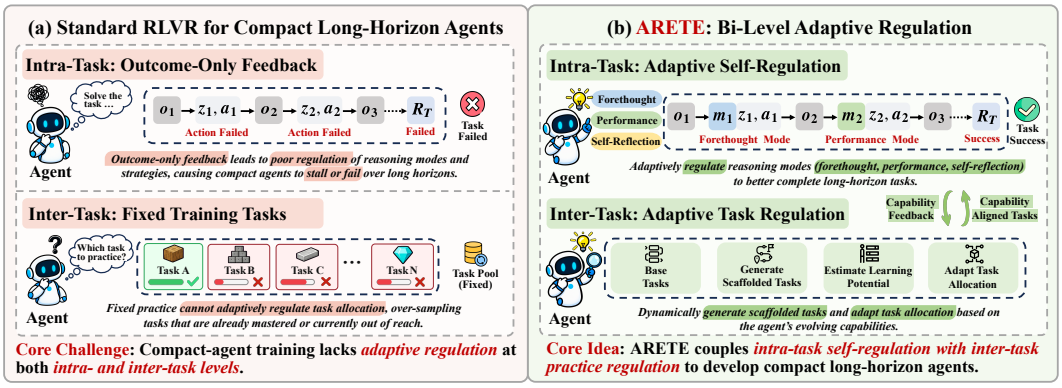


Figure 1: Comparison of (a) outcome-only training with fixed task sampling and (b) ARETE, which combines intra-task self-regulation with capability-matched inter-task practice for compact long-horizon agents.

offers attainable learning opportunities. Effective training of compact agents therefore requires both targeted reasoning feedback and practice matched to their evolving capabilities.

This raises a central research question: ***To improve compact agents' long-horizon performance, how can adaptive regulation align intra-task reasoning with task states, and adapt inter-task practice to evolving capabilities?*** Figure 1 contrasts outcome-only training on fixed tasks with adaptive regulation of both reasoning-mode use and subsequent practice. Addressing this question involves two related challenges. ***First, how can intra-task learning enable agents to self-regulate the selection and switching of reasoning modes according to task demands (How to Act)?*** In human learning, the cyclical model of self-regulated learning (SRL) connects forethought, performance, and self-reflection, emphasizing how learning processes and outcomes inform subsequent strategies Zimmerman (2002); Panadero (2017). This perspective motivates training agents not only to execute these reasoning behaviors, but also to regulate their use based on task progress and environmental feedback. Without such regulation, agents may persist with unsuitable modes and repeat ineffective actions. Differentiated feedback on mode applicability and within-mode execution quality can strengthen this capability, supporting more reliable task completion with fewer unproductive interactions. ***Second, how can inter-task learning construct and select practice tasks that match the agent's evolving capabilities and offer opportunities for improvement (What to Practice)?*** As capabilities evolve, some tasks become mastered, while low-success tasks differ in their learning potential: some become tractable with support, while others remain beyond the agent's current reach. Under a finite sampling budget, this distinction is critical for prioritizing subsequent practice Gaven et al. (2025); Song et al. (2026). The zone of proximal development (ZPD), together with instructional scaffolding, motivates using improvements supported as a proxy for potential learnability Vygotsky & Cole (1978); Wood et al. (1976); Chen et al. (2026). This signal complements current success rates and guides subsequent task construction and selection.

To address these challenges, this paper proposes ARETE (Adaptive REgulation with Tailored Experiences), a reinforcement learning framework that trains compact long-horizon agents through bi-level adaptive regulation of intra- and inter-task learning. At the intra-task level, inspired by the self-regulated learning (SRL) framework from educational psychology Zimmerman (2002); Panadero (2017), ARETE structures reasoning into forethought, performance, and self-reflection, and uses separate, verifiable feedback on mode applicability and execution quality to adapt mode selection and execution. At the inter-task level, inspired by the zone of proximal development from developmental and educational psychology Vygotsky & Cole (1978); Wood et al. (1976); Chen et al. (2026), ARETE generates scaffolded task variants verified for goal consistency and executability, then compares unaided and scaffolded performance to estimate learning potential and guide capability-matched task generation and practice allocation. Together, these mechanisms couple state-dependent reasoning regulation with capability-adaptive practice. Under matched rollout budgets and fixed backbone sizes, experiments on TextCraft and ScienceWorld show that ARETE improves task success rates and reduces redundant actions over competitive methods, enabling a 2B-parameter model to perform competitively with models exceeding 100B parameters.

2 Related Work

2.1 Compact Language Model Agents for Long-Horizon Tasks

Research on compact language model agents seeks to improve long-horizon competence through structured control and targeted training. ReAct and Reflexion establish reasoning–action interleaving and feedback-based refinement, while Voyager develops automatic curricula and reusable skills without parameter updates Yao et al. (2023); Shinn et al. (2023); Wang et al. (2024). LATS and ADaPT use tree search and adaptive task decomposition Zhou et al. (2024a); Prasad et al. (2024). SwiftSage pairs a compact executor with a large-model planner through heuristic switching Lin et al. (2023). Training-based approaches include AutoWebGLM's browsing curricula, AgentGen's synthetic environments and planning tasks, and Agent-R's iterative self-training for error recovery Lai et al. (2024); Hu et al. (2025); Yuan et al. (2025). Learning When to Plan and SR^{2}AM learn when to invoke planning, with SR^{2}AM also regulating planning depth Paglieri et al. (2025); Deng et al. (2026). ARETE instead learns state-dependent regulation among forethought, performance, and self-reflection within a compact policy and couples it with practice adapted to evolving capabilities.

2.2 Reinforcement Learning for Interactive Agents

Reinforcement learning improves interactive agents by optimizing policies through multi-turn environmental feedback. ArCHer and LOOP develop multi-turn policy optimization, whereas RAGEN and AgentGym-RL study rollout design and stable interactive training Zhou et al. (2024b); Chen et al. (2025); Wang et al. (2025); Xi et al. (2026). Agent Lightning v1.0 connects deployment-time agent workflows with RL training He et al. (2026). SCoRe and ARTIST apply RL to self-correction and tool use, respectively, while RLVMR introduces verifiable meta-reasoning rewards and group-relative credit assignment Kumar et al. (2025); Singh et al. (2025); Zhang et al. (2026). Adaptive task construction further complements policy optimization: WebRL generates tasks from unsuccessful attempts, MAGELLAN prioritizes goals using predicted competence and learning progress, GenEnv aligns generated tasks with agent capabilities, and Agent-World integrates capability-gap diagnosis, verifiable task synthesis, and multi-environment RL Qi et al. (2025); Gaven et al. (2025); Guo et al. (2026); Dong et al. (2026). ARETE connects these directions by providing differentiated, verifiable feedback for reasoning-mode selection and execution within tasks, while using differences between unaided and scaffolded performance to guide capability-matched practice across tasks.

3 Preliminaries

Task Definition. Given an initial policy π_{θ_0} and a seed task pool $\mathcal{D}_0$, training proceeds in cycles. At cycle k, a practice set $\mathcal{D}_k$ is constructed by selecting base tasks and forming paired unaided and scaffolded task instances. This set is processed in batches, with multiple rollouts collected per task instance. Interaction feedback guides policy updates and subsequent task selection and generation. The objective is to improve long-horizon task completion through adaptive reasoning regulation and capability-matched practice, without increasing the backbone model size.

Multi-Turn Agent-Environment Interaction. This paper models the multi-turn agent-environment interaction as a Partially Observable Markov Decision Process (POMDP), represented by $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, r)$. Here, $\mathcal{U}$, $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the instruction, state, action, and observation spaces, respectively. $\mathcal{T}$ describes state transitions, and r evaluates task completion. Each task instance $\xi = (x, c)$ specifies an instruction $x \in \mathcal{U}$ and an initial environment configuration c. At turn t, the agent receives an observation o_t and generates $(m_t, z_t, a_t) \sim \pi_\theta(\cdot \mid h_t)$ based on the interaction history $h_t = (x, o_1, m_1, z_1, a_1, \ldots, o_t)$, where m_t, z_t, and a_t denote the reasoning mode, reasoning content, and action, respectively. Only a_t is executed in the environment, which returns the next observation o_{t+1}. Interaction ends upon termination or after $T_{\max}$ turns, yielding a trajectory τ and a task-completion reward $r(\xi, \tau)$.

4 The Proposed ARETE

Overview. ARETE combines intra-task reasoning regulation with inter-task practice regulation to train compact long-horizon agents. As illustrated in Figure 2, intra-task regulation reduces ineffec-

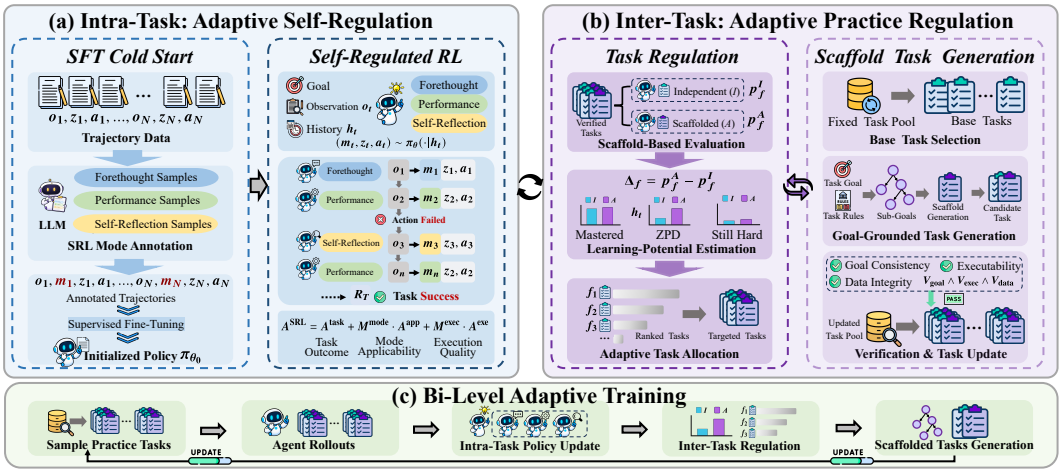


Figure 2: Overview of ARETE. (a) Intra-task regulation learns when and how to use forethought, performance, and self-reflection from applicability and execution feedback. (b) Inter-task regulation estimates learning potential from scaffolded rollouts to guide practice allocation and verified task generation. (c) Bi-level training alternates policy updates, task generation and regulation.

tive trial and error by learning to select and switch among three SRL-inspired modes: *forethought*, *performance*, and *self-reflection*. Supervised fine-tuning establishes the mode-structured reasoning–action interface, while reinforcement learning refines mode selection and execution through separate, verifiable feedback on applicability and execution quality. Inter-task regulation compares paired unaided and scaffolded attempts to estimate potential learnability, guiding verified task generation and practice allocation as capabilities evolve. The resulting interactions update both the policy and subsequent practice, coupling the two levels throughout training.

4.1 Intra-Task Self-Regulated Reinforcement Learning

Inspired by self-regulated learning (SRL), ARETE optimizes reasoning-mode selection and execution to train compact agents to regulate their reasoning in response to environmental feedback. This aims to reduce unproductive trial and error and improve performance on long-horizon tasks.

4.1.1 SRL-Structured Cold-Start Initialization

Self-regulated learning (SRL) describes an adaptive learning process in which preparation, monitoring during performance, and reflection on outcomes inform subsequent strategies. This perspective motivates us to treat reasoning as a process that the agent actively regulates, rather than a fixed behavior repeated throughout an episode. ARETE adapts these regulatory functions into three learnable modes, $\mathcal{M} = \{\mathrm{F}, \mathrm{P}, \mathrm{R}\}$: *forethought* organizes goals and prerequisites, *performance* monitors progress and selects task-directed actions, and *self-reflection* examines unsuccessful attempts and revises subsequent decisions. Each mode produces both reasoning and an action.

The cold-start stage gives the backbone an initial ability to express these functions in structured responses. We use teacher-assisted SRL demonstrations that associate interaction histories with explicit mode labels, reasoning content, and actions. Supervised fine-tuning on these demonstrations yields π_{θ_0}, teaching the policy to generate (m_t, z_t, a_t) and connect each mode with its corresponding reasoning behavior. This provides behavioral examples as well as instruction-following supervision, so subsequent RL can focus on adapting mode use and execution to encountered states instead of learning the output interface from scratch.

4.1.2 Group Relative Policy Optimization for Self-Regulated Learning

Correct execution does not imply that a reasoning mode suits the current task state. ARETE therefore introduces SRL-GRPO, extending GRPO-style optimization (Shao et al., 2024) with separate,

history-calibrated feedback for mode selection and execution. This design trains compact agents to regulate their reasoning as task conditions change, rather than persist with unsuitable modes and ineffective actions. Task completion remains the shared optimization objective. For rollouts sampled from the same prompt group g, let $\mathcal{R}_{i,t,j}$ denote the undiscounted environment return-to-go at token j of turn t in rollout i. The task advantage is $A^{\text{task}}_{i,t,j} = \frac{\mathcal{R}_{i,t,j}-\mu_g}{\sigma_g+\epsilon_{\text{n}}}$, where μ_g and σ_g are computed over valid policy-token positions across the group's rollouts, and ϵ_{n} ensures numerical stability. To provide targeted supervision beyond task outcomes, ARETE constructs two binary signals: mode applicability $v^{\text{app}}_t = V_{\text{app}}(h_t, m_t)$ and execution quality $v^{\text{exe}}_t = V_{m_t}(h_t, z_t, a_t, o_{t+1})$. Applicability is assessed from preceding failures and task progress, whereas execution is assessed through response validity, action legality, and mode-specific behavioral checks. Thus, a valid but unnecessary planning response can receive different feedback for its mode choice and execution. As competence evolves, these signals are calibrated against historical performance:

$$A^{\text{app}}_{i,t} = \omega_u c_{\text{app}} \lambda_u \big(v^{\text{app}}_{i,t} - \widetilde{b}^{\text{app}}_{m_{i,t},u}\big), \qquad A^{\text{exe}}_{i,t} = \omega_u c_{\text{exe}} \big(v^{\text{exe}}_{i,t} - \widetilde{b}^{\text{exe}}_{m_{i,t},u}\big). \tag{1}$$

Here, u indexes policy updates, and $\widetilde{b}^{d}_{m,u}$ is a clipped, mode-specific exponential moving average for feedback type d, updated from unaided rollouts and frozen before sampling. Historical calibration preserves informative feedback even when current labels are identical. The coefficients c_{app} and c_{exe} weight the two signals, while λ_u adjusts selection supervision according to execution competence. The shared scale ω_u caps auxiliary magnitude relative to the task signal, with a small floor allowing auxiliary learning when task advantages vanish. The two auxiliary advantages are assigned to the output segments responsible for the corresponding decisions:

$$A^{\text{SRL}}_{i,t,j} = A^{\text{task}}_{i,t,j} + M^{\text{mode}}_{i,t,j} A^{\text{app}}_{i,t} + M^{\text{exe}}_{i,t,j} A^{\text{exe}}_{i,t}, \tag{2}$$

where the disjoint masks M^{mode} and M^{exe} identify mode-tag tokens and within-mode reasoning/action tokens, respectively. All policy-generated tokens retain task credit. This assignment provides distinct guidance on when to retain or switch a mode and how to carry it out, supporting self-regulation within a shared policy without modifying the environment reward.

Finally, SRL-GRPO maximizes the following clipped objective with advantages held fixed. We suppress rollout, turn, and token indices for clarity:

$$\mathcal{J}_{\text{SRL}}(\theta) = \mathbb{E}_{\text{pol}}\left[\min\left(\rho A^{\text{SRL}}, \text{clip}(\rho, 1-\epsilon_{\text{c}}, 1+\epsilon_{\text{c}}) A^{\text{SRL}}\right)\right] - \beta_{\text{KL}} \overline{D}_{\text{KL}} + \beta_{\text{H}} \overline{\mathcal{H}}. \tag{3}$$

Here, $\rho = \pi_\theta(y \mid \widetilde{h}) / \pi_{\theta_{\text{old}}}(y \mid \widetilde{h})$ is the token-level probability ratio, with $\widetilde{h}$ containing the interaction history and current response prefix. The expectation averages over valid policy-generated tokens sampled under $\pi_{\theta_{\text{old}}}$, excluding observations and padding. ϵ_{c} controls clipping, while $\overline{D}_{\text{KL}}$ and $\overline{\mathcal{H}}$ are the token-averaged reference-policy KL penalty and policy entropy, weighted by β_{KL} and β_{H}. SRL-GRPO therefore retains the GRPO-style update while tailoring its auxiliary credit to reasoning-mode selection and execution.

4.2 Inter-Task Adaptive Practice Regulation

Low task success alone does not establish that further practice will be productive. Inspired by the zone of proximal development and instructional scaffolding (Vygotsky & Cole, 1978; Wood et al., 1976), ARETE uses responsiveness to support to guide task generation and allocation. This shifts practice toward attainable learning opportunities as the agent's capabilities evolve.

4.2.1 Verified Scaffolded Task Generation

Each source task defines a family $f \in \mathcal{F}$ with paired unaided and scaffolded conditions, $\mathcal{P}_f = (\xi^{\text{I}}_f, \xi^{\text{A}}_f)$. Both conditions share the task goal, initial configuration, environment rules, and interaction limit. The scaffolded condition adds instructional guidance on goal decomposition without providing executable commands or additional resources. The same policy snapshot collects G rollouts per condition, providing both training experience and evidence of responsiveness to support.

After warm-up and initial coverage, recent paired outcomes identify candidate parents with intermediate independent success and a positive support gap. A constrained generator produces task variants through bounded edits to task context or initial configuration. To prevent adaptation from compromising task validity, each candidate $\widetilde{\xi}_f$ must pass three checks, summarized as

$\mathcal{V}_f = V_{\text{goal}}(\widetilde{\xi}_f, \xi_f)\, V_{\text{exec}}(\widetilde{\xi}_f)\, V_{\text{data}}(\widetilde{\xi}_f)$. Here, V_{goal} checks goal preservation, V_{exec} verifies executability through environment checks and reference-action replay, and V_{data} checks duplication and held-out exclusions. Only candidates with $\mathcal{V}_f = 1$ are admitted. Accepted variants replace existing practice pairs under a fixed generation cap, with both conditions sharing the modified task. Rejected candidates retain the original pair. Reference actions are used only for verification and are not exposed to the policy.

4.2.2 Support-Responsive Task Regulation

To reduce variability from limited rollouts, ARETE maintains discounted success and failure evidence for each source family and support condition. Each observed condition contributes its cycle-average task success once, while unobserved statistics remain unchanged. The resulting smoothed success estimates are $p^d_{f,k} = \alpha^d_{f,k}/(\alpha^d_{f,k} + \beta^d_{f,k})$ for $d \in \{\mathrm{I}, \mathrm{A}\}$, where α and β accumulate discounted success and failure evidence. These estimates use environment-verified task completion rather than SRL auxiliary feedback. Potential learnability is scored as

$$L_{f,k} = \left[p^{\mathrm{A}}_{f,k} - p^{\mathrm{I}}_{f,k}\right]_+ p^{\mathrm{A}}_{f,k} \left(1 - p^{\mathrm{I}}_{f,k}\right). \tag{4}$$

The three factors capture positive support response, assisted attainability, and remaining independent learning headroom. Their combination downweights both mastered tasks and tasks that remain difficult under available support. This score is a proxy for potential learnability, not a direct estimate of future learning gains. After initial coverage, subsequent practice uses the sampling weights

$$w_{f,k+1} = (1 - \varepsilon_{\mathrm{s}}) \frac{L_{f,k}}{\sum_{f' \in \mathcal{F}} L_{f',k}} + \frac{\varepsilon_{\mathrm{s}}}{|\mathcal{F}|}, \tag{5}$$

where ε_{s} preserves exploration, with uniform sampling if all scores vanish. A fixed number of families are sampled without replacement, each contributing both conditions to $\mathcal{D}_{k+1}$. An admitted verified variant replaces its source pair. Generated-variant outcomes are tracked separately to constrain subsequent synthesis. This provides capability-adaptive experience for intra-task learning.

4.3 Bi-Level Adaptive Training

As illustrated in Figure 2(c), the two levels share interaction evidence but operate at different update frequencies. During cycle k, the current policy collects paired unaided and scaffolded rollouts from $\mathcal{D}_k$ in batches. Each batch supports SRL-GRPO updates, while unaided feedback calibrates auxiliary supervision for subsequent updates. At the cycle boundary, accumulated task outcomes refresh support-response estimates and guide task selection and verified generation for $\mathcal{D}_{k+1}$. Thus, policy development informs subsequent practice, while capability-matched experience supports further learning of reasoning regulation. Practice-set size, rollouts per instance, and interaction limits remain fixed. Task verification adds simulator interactions but no additional policy rollouts. At evaluation, the trained agent autonomously selects reasoning modes on the original held-out tasks without instructional scaffolds.

5 Experiments

This section describes the TextCraft and ScienceWorld benchmarks, training and evaluation protocols, baselines, and metrics. We then conduct experiments to answer the following research questions. The codes are available at `https://anonymous.4open.science/r/ARETE-8FE2`.

Q1: How does ARETE compare with competitive methods in terms of task success across two interactive benchmarks?

Q2: Does ARETE reduce invalid actions and repeated failures compared with competitive methods?

Q3: To what extent do the core components contribute to the performance of ARETE?

Q4: How does ARETE regulate its reasoning modes in response to failed or inconclusive environmental feedback?

Table 1: Evaluation results on TextCraft. Depth 1–4 denote success rates at the corresponding task depths, and Succ. denotes the overall success rate. Light blue and purple bands identify the Qwen3.5-2B and Qwen2.5-3B backbone groups, respectively, while the corresponding darker shading marks ARETE results. The best and second-best results within each compact backbone are shown in **bold** and underlined, respectively. All results are percentages.

Model	Method	Depth 1↑	Depth 2↑	Depth 3↑	Depth 4↑	Succ.↑
Proprietary Models						
GPT-5.6-Luna	ReAct	100.00	100.00	96.00	66.67	98.00
Qwen3.8-Flash	ReAct	100.00	95.12	92.00	66.67	95.00
Qwen3.8-Max	ReAct	100.00	100.00	96.00	100.00	99.00
Open-Source Models (>100B)						
DeepSeek-V4-Flash	ReAct	96.77	95.12	76.00	100.00	91.00
DeepSeek-V4-Pro	ReAct	100.00	95.12	92.00	100.00	96.00
GLM-5.2	ReAct	96.77	100.00	92.00	66.67	96.00
Open-Source Models (<100B)						
Qwen3.8-27B	ReAct	96.77	100.00	32.00	100.00	82.00
Backbone: Qwen3.5-2B						
Qwen3.5-2B	ReAct	<u>74.19</u>	12.20	4.00	0.00	29.00
	+SFT	**100.00**	68.29	20.00	0.00	64.00
	+GRPO	**100.00**	85.37	<u>40.00</u>	<u>33.33</u>	77.00
	ScalingInter-RL	**100.00**	<u>97.56</u>	32.00	0.00	<u>79.00</u>
	ARETE	**100.00**	**100.00**	**68.00**	**100.00**	**92.00**
Backbone: Qwen2.5-3B						
Qwen2.5-3B	ReAct	45.16	9.76	0.00	**0.00**	18.00
	+SFT	<u>93.55</u>	41.46	<u>12.00</u>	**0.00**	49.00
	+GRPO	**100.00**	73.17	<u>12.00</u>	**0.00**	64.00
	ScalingInter-RL	**100.00**	<u>82.93</u>	8.00	**0.00**	<u>67.00</u>
	ARETE	**100.00**	**95.12**	**40.00**	**0.00**	**80.00**

5.1 Experimental Setup

Datasets and Environments. We evaluate agents on TextCraft and ScienceWorld, two text-based environments for multi-step planning and interactive decision-making. TextCraft requires agents to collect materials and execute crafting sequences under Minecraft-style rules. The benchmark includes 374 training tasks and 100 test tasks, categorized into Depth 1-4 according to the minimum crafting-tree depth. ScienceWorld requires agents to manipulate objects and tools through textual commands to complete science tasks. Evaluation is conducted on 200 fixed test instances spanning five categories: Measurement, Conductivity, Object Finding, Chemical Mixing, and Lifespan. Details of data filtering and task distributions are provided in Appendix A.1.

Comparison Methods. We compare three groups of models, as summarized in Tables 1 and 2: (i) proprietary models, including GPT-5.6-Luna, Qwen3.8-Flash, and Qwen3.8-Max; (ii) open-source models with more than 100B parameters, including DeepSeek-V4-Flash, DeepSeek-V4-Pro, and GLM-5.2; and (iii) open-source models with fewer than 100B parameters, including Qwen3.8-27B, Qwen3.5-2B, and Qwen2.5-3B. All standalone LLMs are evaluated under a unified ReAct interaction protocol. For Qwen3.5-2B and Qwen2.5-3B, we compare ReAct, SFT, GRPO, ScalingInter-RL, and ARETE under the same backbone. See Appendix B for training and prompt details.

Evaluation Protocol and Metrics. All methods are evaluated using the same AgentGym interface, with one rollout per test instance and a maximum of 30 interaction steps. We use the success rate (↑) as the primary metric. A TextCraft episode is successful if the agent crafts the target item and receives a reward of 1, whereas a ScienceWorld episode is successful only if its cumulative score reaches 100. For ScienceWorld, we also report the average score (↑), calculated from the maximum cumulative score attained on each trajectory. To evaluate interaction efficiency, we report the invalid action rate (↓) and the repeated failed action rate (↓). See Appendix C for detailed definitions.

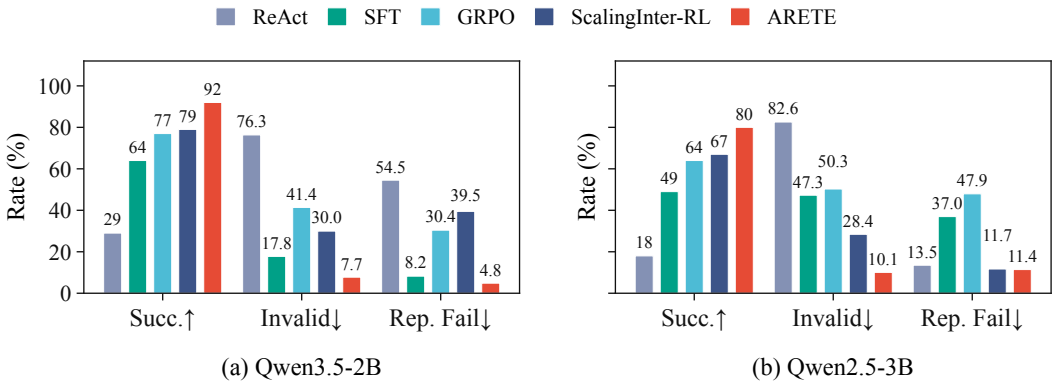
Table 2: Evaluation results on ScienceWorld. Meas., Test-C., Chem., and Life. denote measurement, test conductivity, chemical mixing, and lifespan, respectively. Succ. denotes the overall success rate, and Score denotes the average score. Light blue and purple bands identify the Qwen3.5-2B and Qwen2.5-3B backbone groups, respectively, while the corresponding darker shading marks ARETE results. The best and second-best results within each compact backbone are shown in **bold** and underlined, respectively. All metrics except Score are percentages.

Model	Method	Meas.↑	Test-C.↑	Find↑	Chem.↑	Life.↑	Succ.↑	Score↑
Proprietary Models								
GPT-5.6-Luna	ReAct	66.04	48.19	68.18	60.00	100.00	61.50	84.02
Qwen3.8-Flash	ReAct	52.83	14.46	43.18	40.00	66.67	35.50	64.96
Qwen3.8-Max	ReAct	49.06	19.28	68.18	20.00	93.33	43.50	76.68
Open-Source Models (>100B)								
DeepSeek-V4-Flash	ReAct	54.72	12.05	50.00	0.00	73.33	36.00	63.58
DeepSeek-V4-Pro	ReAct	54.72	12.05	61.36	20.00	33.33	36.00	75.46
GLM-5.2	ReAct	79.25	8.43	77.27	40.00	86.67	49.00	83.11
Open-Source Models (<100B)								
Qwen3.8-27B	ReAct	35.85	10.84	38.64	0.00	93.33	29.50	62.50
Backbone: Qwen3.5-2B								
Qwen3.5-2B	ReAct	5.66	2.41	0.00	0.00	0.00	2.50	2.88
	+SFT	0.00	0.00	52.27	0.00	80.00	17.50	62.33
	+GRPO	33.96	13.25	56.82	**20.00**	80.00	33.50	54.87
	ScalingInter-RL	35.85	14.46	59.09	**20.00**	86.67	35.50	56.24
	ARETE	**45.28**	**56.63**	**95.45**	**20.00**	**100.00**	**64.50**	**72.94**
Backbone: Qwen2.5-3B								
Qwen2.5-3B	ReAct	0.00	0.00	2.27	0.00	0.00	0.50	4.46
	+SFT	9.43	1.20	13.64	0.00	6.67	6.50	45.26
	+GRPO	22.64	**9.64**	22.73	0.00	0.00	15.00	48.42
	ScalingInter-RL	15.09	3.61	43.18	0.00	**60.00**	19.50	50.36
	ARETE	**30.19**	7.23	**56.82**	**20.00**	**60.00**	**28.50**	**61.14**

5.2 Experimental Results and Analysis

Main Performance Evaluation (To Q1). To assess whether bi-level regulation of intra-task reasoning and inter-task practice translates into robust long-horizon performance, we compare ARETE with competitive methods across two environments and compact backbones. As shown in Tables 1 and 2, ARETE achieves the highest overall success rate for both backbones. In TextCraft, it improves the strongest baseline from 79.00% to 92.00% with Qwen3.5-2B and from 67.00% to 80.00% with Qwen2.5-3B; both backbones also gain 28 percentage points in Depth 3 tasks. In ScienceWorld, ARETE raises the success from 35.50% to 64.50% and from 19.50% to 28.50%, while improving the average score by 10.61 and 10.78 points, respectively. Beyond compact-model baselines, Qwen3.5-2B with ARETE reaches 92.00% on TextCraft, approaching the 91.00–99.00% range of much larger models, and attains the highest overall success rate on ScienceWorld. These results suggest that regulating when and how reasoning modes are used within tasks, together with which tasks are generated and prioritized across training, enables compact agents to use their fixed capacity more effectively and substantially narrows the performance gap with larger models.

Interaction Efficiency Analysis (To Q2). We compare ARETE with ReAct, SFT, GRPO, and ScalingInter-RL to assess whether its higher task success is accompanied by more efficient interaction rather than repeated trial and error. Figure 3 shows that ARETE achieves the lowest rates of invalid-action and repeated-failure on both backbones. These rates decrease from the best baseline results of 17.8% / 8.2% to 7.7% / 4.8% in Qwen3.5-2B and from 28.4% / 11.7% to 10.1% / 11.4% in Qwen2.5-3B. ARETE also improves the performance of the task by 13% over the strongest baseline in both backbones. These concurrent gains indicate that ARETE improves interaction efficiency rather than simply shifting failures between invalid actions and repeated attempts, consistent with its differentiated feedback on mode applicability and execution quality.



(a) Qwen3.5-2B (b) Qwen2.5-3B

Figure 3: Task success and interaction efficiency on TextCraft. Succ., Invalid, and Rep. Fail denote success, invalid-action, and repeated-failure rates, respectively.

Ablation Study (To Q3). To quantify the contributions of ARETE's core components, we construct four ablated variants by removing the SRL cold start, replacing SRL-GRPO with standard outcome-level GRPO, removing the complete intra-task regulation module, or replacing inter-task regulation with fixed-task training. As shown in Table 3, all variants underperform the full model on both benchmarks. Removing intra-task regulation causes the largest drops, reducing success by 12.0 and 10.0 percentage points on TextCraft and ScienceWorld, respectively. Within this module, replacing SRL-GRPO produces larger reductions than removing the cold start, highlighting the importance of differentiated feedback for mode selection and execution. Fixed-task training reduces success by 4.0 and 3.0 points, confirming the benefit of adaptive practice.

Table 3: Component ablation of ARETE using Qwen2.5-3B. Values are success rates (%).

Variant	TextCraft	ScienceWorld
ARETE (Full)	**80.0**	**28.5**
w/o SRL Cold Start	78.0	26.0
w/o SRL-GRPO	71.0	20.5
w/o Intra-Task Regulation	68.0	18.5
w/o Inter-Task Regulation	76.0	25.5

Reasoning-Mode Regulation Analysis (To Q4). To examine reasoning-mode regulation during interaction, we compare ARETE with ScalingInter-RL in one TextCraft and one ScienceWorld case. As detailed in Appendix F, failed or inconclusive feedback prompts ARETE to switch from performance mode to self-reflection mode, revise its strategy, and return to performance mode once a corrective path is identified; ScalingInter-RL instead persists with an unproductive strategy. These cases qualitatively illustrate mode switching in response to environmental feedback and complement the aggregate efficiency results.

6 Conclusion

This paper introduces ARETE, a bi-level reinforcement learning framework that jointly regulates reasoning within tasks and practice across tasks for compact long-horizon agents. Within tasks, ARETE organizes reasoning into forethought, performance, and self-reflection, using differentiated, verifiable feedback to regulate mode selection and execution. Across tasks, it estimates learning potential from unaided and scaffolded performance to guide verified scaffolded task generation and capability-matched practice. Experiments across two complex multi-turn interactive benchmarks show consistent task-success gains across two compact backbones, enabling a 2B model to perform competitively with models exceeding 100B parameters. Interaction analysis further shows fewer invalid actions and repeated failures, while ablations confirm the complementary contributions of both regulation levels. By improving compact agents without increasing backbone size, ARETE offers a general route to reducing the token-processing and computational costs associated with repeated calls to large external models. Future work will scale ARETE to longer and more complex task sequences, improve training and inference efficiency, and investigate local training and deployment in domains involving personal or proprietary data, including education, healthcare, finance, and enterprise services.

AI USE STATEMENT

Generative AI tools were used for translation and manuscript editing. All AI-assisted content was reviewed and verified by the authors.

REPRODUCIBILITY STATEMENT

Method details, experimental settings, data splits, training procedures, prompts, and evaluation protocols are provided in the main text and Appendix. Training budgets, random seeds, and implementation details are also documented. Code and data are available through the repository referenced in Section 5.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Kevin Chen, Marco F. Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive LLM agents. *CoRR*, abs/2502.01600, 2025.

Xuanzhong Chen, Zile Qiao, Guoxin Chen, Liangcai Su, Zhen Zhang, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Yong Jiang, and Ting Chen. Expanding the capability frontier of LLM agents with ZPD-guided data synthesis. In *International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026.

Mingkai Deng, Jinyu Hou, Lara Sá Neves, Varad Pimpalkhute, Taylor W. Killian, Zhengzhong Liu, and Eric P. Xing. Efficient agentic reasoning through self-regulated simulative planning. *CoRR*, abs/2605.22138, 2026.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *CoRR*, abs/2604.18292, 2026.

Loris Gaven, Thomas Carta, Clément Romac, Cédric Colas, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Magellan: Metacognitive predictions of learning progress guide autotelic llm agents in large goal spaces. *arXiv preprint arXiv:2502.07709*, 2025.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. GenEnv: Difficulty-aligned co-evolution between LLM agents and environment simulators. In *International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K. Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1.0: Towards harnessed agentic RL. *CoRR*, abs/2608.17528, 2026.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In Yizhou Sun, Flavio Chierichetti, Hady W. Lauw, Claudia Perlich, Wee Hyong Tok, and Andrew Tomkins (eds.), *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 496–507, Toronto, ON, Canada, 2025. ACM.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, Singapore, 2025. OpenReview.net.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In Ricardo Baeza-Yates and Francesco Bonchi (eds.), *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, Barcelona, Spain, 2024. ACM.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023*, New Orleans, LA, 2023.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. SPIRAL: self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *CoRR*, abs/2506.24119, 2025.

Davide Paglieri, Bartlomiej Cupial, Jonathan Cook, Ulyana Piterbarg, Jens Tuyls, Edward Grefenstette, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. Learning when to plan: Efficiently allocating test-time compute for LLM agents. *CoRR*, abs/2509.03581, 2025.

Ernesto Panadero. A review of self-regulated learning: Six models and four directions for research. *Frontiers in psychology*, 8:422, 2017.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, Mexico City, Mexico, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations,*, Singapore, 2025. OpenReview.net.

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. *CoRR*, abs/2402.03300, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023*, New Orleans, LA, 2023.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. *CoRR*, abs/2505.01441, 2025.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026*, pp. 8326–8357, San Diego, CA, 2026.

Georgios Tzannetos, Bárbara Gomes Ribeiro, Parameswaran Kamalaruban, and Adish Singla. Proximal curriculum for reinforcement learning agents. *Trans. Mach. Learn. Res.*, 2023.

Lev Semenovich Vygotsky and Michael Cole. *Mind in society: Development of higher psychological processes*. Harvard university press, 1978.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024, 2024.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: understanding self-evolution in LLM agents via multi-turn reinforcement learning. *CoRR*, abs/2504.20073, 2025.

David Wood, Jerome S Bruner, and Gail Ross. The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, 17(2):89–100, 1976.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evaluating and training large language model-based agents across diverse environments. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 27914–27961, Vienna, Austria, 2025. Association for Computational Linguistics.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Jiaqi Liu, Honglin Guo, Yajie Yang, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. Agentgym-rl: An open-source framework to train llm agents for long-horizon decision making via multi-turn rl. In *International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, Kigali, Rwanda, 2023. OpenReview.net.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *CoRR*, abs/2501.11425, 2025.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents. In *COLM*, 2024.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. In *International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, Vienna, Austria, 2024a.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*, volume 235, pp. 62178–62209, Vienna, Austria, 2024b.

Barry J Zimmerman. Becoming a self-regulated learner: An overview. *Theory into practice*, 41(2): 64–70, 2002.

A Datasets and Task Splits

A.1 Dataset Statistics

Table 4 summarizes the task pools, fixed test sets, and SRL-SFT supervision data for both environments. We construct 1,149 single-step supervision samples per environment, covering forethought, performance, and self-reflection. These samples are drawn from 247 source tasks in TextCraft and 206 in ScienceWorld, with SFT source task IDs disjoint from the corresponding test task IDs.

Table 4: Dataset sizes and supervision statistics for TextCraft and ScienceWorld.

Statistic	TextCraft	ScienceWorld
Original training tasks / task pool	374	2,120
Fixed test tasks	100	200
SRL-SFT single-step supervision samples	1,149	1,149
Source task IDs for SFT	247	206
Overlap between SFT source IDs and test IDs	0	0
Forethought samples	209	287
Performance samples	539	575
SelfReflection samples	401	287

A.2 Difficulty and Category Distributions

TextCraft. Table 5 reports the training and test distributions across crafting depths. Task depth is the minimum crafting-tree depth of the target item. Base materials have depth 0, and a recipe has depth one plus the maximum depth of its required ingredients; the minimum is taken over alternative recipes for the same item. Thus, depth captures recipe dependencies rather than the minimum number of interaction steps.

Table 5: Distribution of training and test tasks across task depths in TextCraft.

Task Depth	Training Tasks	Test Tasks
Depth 1	93	31
Depth 2	233	41
Depth 3	47	25
Depth 4	1	3
Total	**374**	**100**

ScienceWorld. Table 6 summarizes the 200 fixed test instances across measurement, conductivity testing, object finding, chemical mixing, and lifespan tasks. Category-specific success rates are computed within the corresponding subsets, whereas the overall success rate is computed across all 200 instances rather than averaged across categories.

Table 6: Distribution of test tasks across ScienceWorld categories.

ScienceWorld Category	Number of Tasks
Measure	53
Test-Cond.	83
Find	44
Chem-Mix	5
Lifespan	15
Total	**200**

A.3 Environment Interfaces and Illustrative Tasks

This section presents the task specifications of TextCraft and ScienceWorld.

A.3.1 TextCraft: Compositional Crafting

TextCraft is a text-based crafting environment in which an agent receives a target item and task-specific crafting recipes. Using the `get`, `inventory`, and `craft` commands, the agent acquires basic ingredients, inspects its inventory, and combines materials to produce intermediate and target items. Task completion requires following recipe dependencies and tracking the types and quantities of available materials. The following example illustrates these requirements through a wooden-pickaxe crafting task.

Task:
Craft a wooden pickaxe.

Initial inventory:
Empty.

Relevant crafting recipes:

- `craft 4 oak planks using 1 oak log`
- `craft 4 sticks using 2 oak planks`
- `craft 1 wooden pickaxe using 3 oak planks and 2 sticks`

Action 1:
`get 2 oak logs`

Observation 1:
Two oak logs are now available.

Action 2:
`craft 4 oak planks using 1 oak log`

Observation 2:
Four oak planks have been produced.
One oak log remains.

Action 3:
`craft 4 oak planks using 1 oak log`

Observation 3:
Four additional oak planks have been produced.
Eight oak planks are now available.

Action 4:
`craft 4 sticks using 2 oak planks`

Observation 4:
Four sticks have been produced.
Six oak planks remain.

Action 5:
`inventory`

Observation 5:
```
oak planks:  6
sticks:  4
```

Action 6:
`craft 1 wooden pickaxe using 3 oak planks and 2 sticks`

Observation 6:
The wooden pickaxe has been crafted successfully.
Three oak planks and two sticks remain.

Illustrated outcome:
The target item is successfully crafted and the task is completed.

A.3.2 SCIENCEWORLD: INTERACTIVE SCIENTIFIC TASKS

ScienceWorld is a text-based environment in which agents solve scientific tasks through sequential interactions with objects and experimental apparatus. Given a task description and textual observations, an agent navigates rooms, conducts experiments, and uses environmental feedback to guide subsequent actions. In the electrical-conductivity task below, the agent inserts a metal fork into a test circuit, inspects the bulb's response to determine whether the fork conducts electricity, and places it in the corresponding answer box.

Task:
Determine whether the metal fork is electrically conductive. First, focus on the fork. Place it in the blue box if it conducts electricity, or in the orange box otherwise.

Initial state:
The agent is in the workshop. A battery, a red light bulb, and three wires form a test circuit with a gap for the sample. The two free contacts are `green wire terminal 2` and `orange wire terminal 1`. The bulb is off. The metal fork and both answer boxes are accessible.

Relevant interaction commands:

- `focus on OBJ`
- `connect OBJ to OBJ`
- `wait`
- `look at OBJ`
- `move OBJ to OBJ`

Action 1:
`focus on metal fork`
Observation 1:
The metal fork is now the focus of the task.

Action 2:
`connect metal fork terminal 1 to green wire terminal 2`
Observation 2:
The first terminal of the fork is connected to the green wire.

Action 3:
`connect metal fork terminal 2 to orange wire terminal 1`
Observation 3:
The second terminal of the fork is connected to the orange wire.
The fork now bridges the gap in the test circuit.

Action 4:
`wait`
Observation 4:
Time passes, allowing the circuit to respond.

Action 5:
`look at red light bulb`
Observation 5:
The red light bulb is on.

Action 6:
`move metal fork to blue box`
Observation 6:
The metal fork is placed in the blue box.

Outcome:
The illuminated bulb provides evidence that the fork conducts electricity. The fork is placed in the corresponding answer box, completing the task.

B MODELS, PROMPT FORMATS, AND TRAINING HYPERPARAMETERS

B.1 MODELS AND OUTPUT FORMATS

We conduct comparisons at two levels. First, for general-purpose LLMs, we evaluate Qwen3.8-Flash, Qwen3.8-Max, DeepSeek-V4-Flash, DeepSeek-V4-Pro, GLM-5.2, Qwen3.8-27B, and GPT-5.6-Luna using a unified ReAct interaction format. Second, for controlled comparisons under the same backbone, We use Qwen3.5-2B and Qwen2.5-3B-Instruct (Qwen2.5-3B) to compare ReAct, SFT, GRPO, ScalingInter-RL, and ARETE.

The ReAct output format is:

```
Thought:
<reasoning>

Action:
<next action>
```

The SRL-structured output additionally includes an explicit self-regulated learning phase:

```
[Performance_mode]
Reasoning: <reasoning>
Action: <next action>
```

The phase label is selected from `[Forethought_mode]`, `[Performance_mode]`, and `[SelfReflection_mode]`.

B.2 SRL-STRUCTURED SUPERVISED FINE-TUNING

We perform supervised fine-tuning (SFT) with an explicit Self-Regulated Learning (SRL) phase structure, enabling the model to distinguish among forethought, performance, and self-reflection during task interaction. Training is initialized from the original Qwen3.5-2B and Qwen2.5-3B-Instruct checkpoints without prior task-specific fine-tuning. For each environment, we use 1,149 single-step supervision samples, as described in Appendix A. Each target response contains an SRL phase label, a reasoning trace, and an action.

For Qwen3.5-2B, all parameters in the language component are updated, while the vision encoder and feature-fusion module are frozen. For Qwen2.5-3B-Instruct, all model parameters are updated. Neither LoRA nor quantization is used.

Training minimizes the autoregressive cross-entropy loss. The loss is computed only over the final model response and its end-of-sequence token, while tokens belonging to the input context and padding positions are masked out.

Both models use the same core training configuration. The training budget is set to 5 epochs. We use the AdamW optimizer with an initial learning rate of 1×10^{-5}, cosine learning-rate decay, a warmup ratio of 0.03, and zero weight decay. The per-device batch size is 1 single-step sample, and gradient accumulation is used to obtain a global effective batch size of 32 samples. The maximum combined input-output sequence length is 4,096 tokens. Training is performed in BF16 precision. Both the training seed and data-shuffling seed are fixed to 11.

Training follows the predefined budget, and the final checkpoint saved at the end of each run is used for evaluation. The realized number of training epochs and optimization steps for each run is reported separately where applicable.

B.3 REINFORCEMENT LEARNING SETTINGS

All reinforcement-learning baselines are implemented using the AgentGym-RL framework, with environment-specific training budgets and interaction horizons for TextCraft and ScienceWorld. To maintain consistent optimization settings, GRPO and ScalingInter-RL on TextCraft and GRPO on ScienceWorld use AdamW with a fixed learning rate of 1×10^{-6}, a policy clipping threshold of

0.2, a gradient-norm clipping threshold of 1.0, and a KL regularization coefficient of 0.001. The sampling temperature and top_p are both set to 1 during training.

For TextCraft, GRPO and ScalingInter-RL are initialized from the original Qwen3.5-2B checkpoint. All language-model parameters are updated, while the vision-related modules are frozen. Both methods use the same set of 374 training tasks. Each sampling batch contains 32 tasks, with 8 trajectories generated for each task. Training is conducted for 10 epochs. Incomplete batches are dropped at the end of each epoch, yielding 11 sampling batches per epoch and 110 in total, with a trajectory-sampling budget of 28,160 trajectories. Each model response is limited to 512 generated tokens, and the maximum context length is 32,768 tokens. The random seed is fixed to 11.

GRPO and ScalingInter-RL use the same training-task set and trajectory-sampling budget, while retaining their respective interaction schedules and optimization procedures. GRPO uses a fixed maximum interaction horizon of 30 rounds throughout training. ScalingInter-RL progressively increases the maximum interaction horizon: 10 rounds for sampling batches 1–33, 20 rounds for batches 34–66, and 30 rounds for batches 67–110. Thus, the two methods use the same trajectory-sampling budget, while the actual numbers of environment interactions and generated tokens are not constrained to be identical.

B.4 Agent Prompt Templates and Interaction Protocols

This section presents the fixed SRL-structured and ReAct prompts used in TextCraft and ScienceWorld. The SRL-structured prompts require the agent to select exactly one reasoning mode from `[Forethought_mode]`, `[Performance_mode]`, and `[SelfReflection_mode]` before producing task-relevant reasoning and a single executable action. In contrast, the ReAct prompts follow a standard Thought and Action format without an explicit reasoning mode tag.

B.4.1 SRL-Structured Agent Prompts

TextCraft.

```
You are a TextCraft agent using self-regulated learning.
Crafting commands have the format "craft [target object] using [input ingredients]".

At each step, choose exactly one SRL reasoning mode:
[Forethought_mode] for planning a new, complex, or unclear task.
[Performance_mode] when the next useful action is clear.
[SelfReflection_mode] after a failed action, stalled progress, or unchanged state.

Your response MUST start with exactly one of:
[Forethought_mode]
[Performance_mode]
[SelfReflection_mode]

Then output exactly:
Reasoning: brief task-relevant reasoning.
Action: one executable TextCraft command

Any response without exactly one reasoning-mode tag is invalid.
Any response without Reasoning is invalid.
Any response without Action is invalid.

Valid actions are only:
- get [quantity] [item]
- craft [quantity] [target] using [ingredients]
- inventory

Rules:
1. Always specify quantity for get and craft commands.
2. get commands must NEVER contain "using".
3. inventory must not include any arguments.
4. Use craft commands exactly as provided, except generic ingredients such as planks may use a
 specific type only when the command allows it.
5. Do not output WAIT, done, stop, finish, or multiple actions.

Invalid:
Action: craft 1 piston using 1 redstone, 4 cobblestone, 3 planks, 1 iron ingot

Valid:
[Performance_mode]
```

```
Reasoning: I have all required ingredients for piston.
Action: craft 1 piston using 1 redstone, 4 cobblestone, 3 oak planks, 1 iron ingot

Only the command after Action: will be sent to the environment.
```

ScienceWorld.

```
You are an agent for the ScienceWorld environment. Every round I will give you an observation,
 and you have to respond with an action based on the observation to finish the given task.

Here are the actions you may take:

{"action": "open/close OBJ", "description": "open/close a container"}
{"action": "de/activate OBJ", "description": "activate/deactivate a device"}
{"action": "connect OBJ to OBJ", "description": "connect electrical components"}
{"action": "disconnect OBJ", "description": "disconnect electrical components"}
{"action": "use OBJ [on OBJ]", "description": "use a device/item"}
{"action": "look around", "description": "describe the current room"}
{"action": "look at OBJ", "description": "describe an object in detail"}
{"action": "look in OBJ", "description": "describe a container's contents"}
{"action": "read OBJ", "description": "read a note or book"}
{"action": "move OBJ to OBJ", "description": "move an object to a container"}
{"action": "pick up OBJ", "description": "move an object to the inventory"}
{"action": "put down OBJ", "description": "drop an inventory item"}
{"action": "pour OBJ into OBJ", "description": "pour a liquid into a container"}
{"action": "dunk OBJ into OBJ", "description": "dunk a container into a liquid"}
{"action": "mix OBJ", "description": "chemically mix a container"}
{"action": "go to LOC", "description": "move to a new location"}
{"action": "eat OBJ", "description": "eat a food"}
{"action": "flush OBJ", "description": "flush a toilet"}
{"action": "focus on OBJ", "description": "signal intent on a task object"}
{"action": "wait", "description": "take no action for 10 iterations"}
{"action": "wait1", "description": "take no action for 1 iteration"}
{"action": "task", "description": "describe the current task"}
{"action": "inventory", "description": "list your inventory"}

Choose one reasoning mode based only on the task, current observation, and visible history.
Choose the mode for the purpose of the current decision, not for the action verb.

Forethought: establish or revise a plan for an unresolved multi-step goal.
At the start of an unfamiliar multi-stage task, briefly identify the necessary
subgoals or ordering and execute the first justified step. A planning decision
can execute an observation, navigation, or focus action. Do not label a plan
Performance merely because its first action is simple.

Performance: carry out a clear next step in an established plan, including useful
observation or waiting. A genuinely one-step initial decision can also use this mode.

SelfReflection: diagnose a real, still-unresolved failure or evidence conflict
visible in the history, then execute a corrective or diagnostic action. Once
the diagnosis is settled and the next step is clear, resume Performance; a new
overall strategy can use Forethought. Do not keep reflecting just because an
earlier error exists somewhere in the history.

For example, deciding how to obtain ingredients, combine them, and complete a
goal is Forethought; opening a known container as the next step of that plan
is Performance; investigating a failed interaction is SelfReflection. These
are mode-selection examples, not claims about objects in the current task.

Do not force every first step to Forethought or follow a fixed mode schedule.
Zero score increase alone does not mean failure. Never invent observations.

The feedback "No known action matches that input." means the action was rejected.
It does not identify the cause: missing objects, naming, and prerequisites must
be checked against visible evidence. State an uncertain cause as a possibility.

Use task-stated locations and observed exits. Explain an intermediate room as
a route step when relevant. Do not invent routes or presume unrelated containers
hold task materials. The action must implement the explanation.

Return exactly three lines, without Markdown fences or extra text:
[Forethought_mode] OR [Performance_mode] OR [SelfReflection_mode]
Reasoning: one concise explanation, at most 50 words, on one line.
Action: exactly one ScienceWorld action from the action syntax above.

Use exactly ONE of the three reasoning-mode tags on the first line.
```

B.4.2 REACT AGENT PROMPTS

TextCraft.

```
You are an agent interacting with the TextCraft environment. The environment
will provide a goal and a set of useful Minecraft crafting recipes

At each round, you will receive an observation describing the available
crafting commands, the goal, your inventory, or the result of your previous
action. Select one action that moves you toward completing the goal.

You may perform only the following actions:
1. Get an item:
   get [quantity] [item]
2. Craft an item with one of the recipes provided in the observation:
   craft [quantity] [target item] using [quantity] [ingredient], [quantity] [ingredient], ...
3. Inspect the current inventory:
   inventory
Rules:
1. Always include a quantity in every get or craft action.
2. A get action must contain only the quantity and item name. Do not state
   whether the item comes from the environment or inventory.
3. Use only crafting recipes supplied by the environment. Do not invent recipes.
4. Preserve the ingredient quantities specified by the supplied recipe.
5. If a supplied recipe uses a generic ingredient, such as planks, a compatible
   specific type, such as dark oak planks, may be used.
6. Output exactly one action in each round.
7. Do not output multiple Action fields.
8. Do not output done, finish, stop, wait, or any action outside the three
   supported action types.
9. The environment determines whether the task has been completed.

Your response must use exactly the following format:
Thought:
A brief explanation of the next step.
Action:
One executable TextCraft command
```

ScienceWorld.

```
You are an agent for the ScienceWorld environment. Every round I will give you an observation,
 and you have to respond with an action based on the observation to finish the given task.

Here are the actions you may take:
{"action": "open/close OBJ", "description": "open/close a container"}
{"action": "de/activate OBJ", "description": "activate/deactivate a device"}
{"action": "connect OBJ to OBJ", "description": "connect electrical components"}
{"action": "disconnect OBJ", "description": "disconnect electrical components"}
{"action": "use OBJ [on OBJ]", "description": "use a device/item"}
{"action": "look around", "description": "describe the current room"}
{"action": "look at OBJ", "description": "describe an object in detail"}
{"action": "look in OBJ", "description": "describe a container's contents"}
{"action": "read OBJ", "description": "read a note or book"}
{"action": "move OBJ to OBJ", "description": "move an object to a container"}
{"action": "pick up OBJ", "description": "move an object to the inventory"}
{"action": "put down OBJ", "description": "drop an inventory item"}
{"action": "pour OBJ into OBJ", "description": "pour a liquid into a container"}
{"action": "dunk OBJ into OBJ", "description": "dunk a container into a liquid"}
{"action": "mix OBJ", "description": "chemically mix a container"}
{"action": "go to LOC", "description": "move to a new location"}
{"action": "eat OBJ", "description": "eat a food"}
{"action": "flush OBJ", "description": "flush a toilet"}
{"action": "focus on OBJ", "description": "signal intent on a task object"}
{"action": "wait", "description": "take no action for 10 iterations"}
{"action": "wait1", "description": "take no action for 1 iteration"}
{"action": "task", "description": "describe the current task"}
{"action": "inventory", "description": "list your inventory"}

Your response should use the following format:
Thought:
your thoughts.
Action:
your next action
```

Table 7: Operational definitions of the evaluation metrics and success criteria.

Metric / Criterion	Operational Definition
TextCraft success	A task is considered successful when `task_reward >= 1.0`. Task reward is used as the primary success criterion; `done` alone is insufficient.
ScienceWorld success	A task is considered successful only when its cumulative score reaches 100 within the interaction limit. Environment termination alone or a positive partial score does not constitute success.
ScienceWorld average score	For each task, the maximum cumulative score attained during the trajectory is recorded and then averaged across all test instances. This metric captures partial task progress independently of binary task success.
Invalid action	The union of format-invalid actions and actions rejected by the environment. Each model decision contributes at most one invalid-action count.
Repeated failed action	An action is counted only when it repeats the immediately preceding failed action after action normalization. Repetitions of arbitrary actions from earlier interaction history are not included. The repeated failed action rate (RFAR) is computed over post-failure opportunities in which a subsequent model decision is actually made; a failed final action without a subsequent decision contributes no opportunity.

Note. Ratio-based metrics use globally aggregated numerators and denominators rather than averages of per-task percentages. Depth- and category-specific success rates are computed within the corresponding subsets. When the aggregated number of post-failure opportunities is zero, the evaluation implementation records RFAR as 0 while retaining the corresponding opportunity count.

C Evaluation Metrics and Success Criteria

Let N denote the number of evaluated tasks, s_i the success indicator for task i, T_i the number of model decisions, u_i the number of invalid-action attempts, f_i the number of realized decision opportunities immediately following a failed action, and r_i the number of such opportunities in which the model selects an action that is identical, after normalization, to the immediately preceding failed action.

The success rate (SR), invalid action rate (IAR), and repeated failed action rate (RFAR) are defined as

$$\mathrm{SR} = \frac{\sum_{i=1}^{N} s_i}{N}, \qquad \mathrm{IAR} = \frac{\sum_{i=1}^{N} u_i}{\sum_{i=1}^{N} T_i}, \qquad \mathrm{RFAR} = \frac{\sum_{i=1}^{N} r_i}{\sum_{i=1}^{N} f_i}. \tag{6}$$

The RFAR ratio applies when $\sum_{i=1}^{N} f_i > 0$; when this denominator is zero, the evaluation implementation records RFAR as 0 while retaining the corresponding opportunity count.

For ScienceWorld, the average score is computed as

$$\mathrm{AverageScore} = \frac{1}{N} \sum_{i=1}^{N} \max_{t} S_{i,t}, \tag{7}$$

where $S_{i,t}$ denotes the cumulative environment score for task i at interaction step t. This metric averages the maximum cumulative score reached along each task trajectory rather than the final score alone or the sum of per-step reward increments.

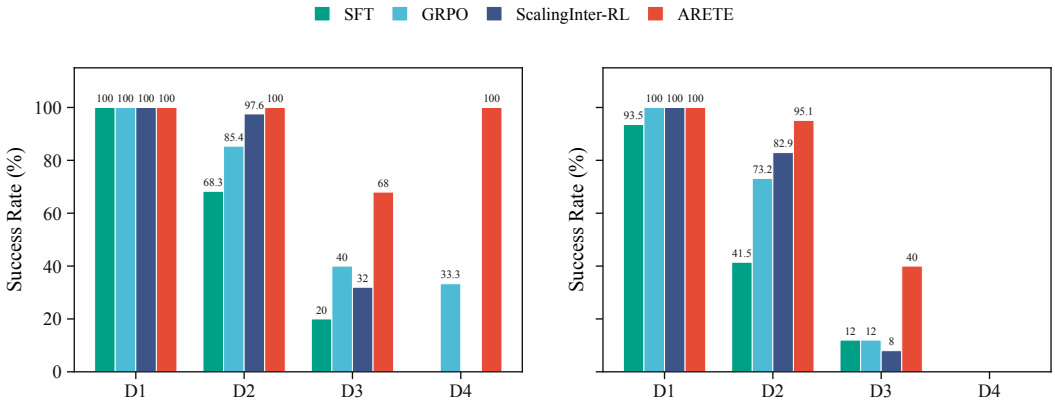
The operational definitions and aggregation rules used in evaluation are summarized in Table 7.

D Experimental Environment

All experiments are conducted on a server equipped with four NVIDIA H100 80 GB GPUs. The hardware and software configurations are summarized in Table 8.

Table 8: Hardware and software configurations used in the experiments.

Category	Component	Configuration
Hardware	GPU	4 × NVIDIA H100 80 GB
Software	Python	3.10
	PyTorch	2.4.0
	CUDA	12.1
	Transformers	5.4.0
	Accelerate	1.13.0
	DeepSpeed	0.18.9
	Datasets	4.8.4
	FlashAttention	2.8.3



(a) Qwen3.5-2B (b) Qwen2.5-3B

Figure 4: TextCraft success rates across task depths. Depth 4 contains only three test tasks and should be interpreted cautiously.

E Additional Experimental Analyses

E.1 Performance across Crafting Depths

Figure 4 reports TextCraft success rates at different crafting depths. Performance on Depth 1 is largely saturated across training methods, whereas the differences become more pronounced as the prerequisite structure deepens. On Qwen3.5-2B, ARETE improves Depth 3 success from the strongest baseline result of 40.0% to 68.0%. On Qwen2.5-3B, it improves Depth 2 and Depth 3 success to 95.1% and 40.0%, respectively. These results indicate that ARETE is particularly beneficial when task completion requires reasoning over longer dependency chains. Because Depth 4 contains only three test instances, its results should be interpreted cautiously.

E.2 Additional Backbone Evaluation

We further evaluate ARETE with Qwen3.5-9B on the same 100 TextCraft test tasks under the 30-step interaction limit. Table 9 compares the three evaluated Qwen backbones. Within the Qwen3.5 family, increasing the backbone from 2B to 9B improves overall success from 92.0% to 95.0%, while reducing the invalid-action rate from 7.70% to 1.85% and the repeated-failure rate from 4.80% to 2.47%. The largest depth-wise improvement occurs on Depth 3, where success increases from 68.0% to 88.0%. Depth 4 does not follow the same trend, but this subset contains only three instances. Because only ARETE is evaluated on Qwen3.5-9B, this experiment provides additional backbone evidence rather than a controlled same-backbone comparison with other training methods.

Table 9: ARETE results across Qwen backbones on TextCraft. Depth1–Depth4 denote depth-specific success rates; Invalid and Rep. Fail denote invalid-action and repeated-failure rates. All results are percentages.

Backbone	Depth1↑	Depth2↑	Depth3↑	Depth4↑	Succ.↑	Invalid↓	Rep. Fail↓
Qwen2.5-3B	**100.00**	95.12	40.00	0.00	80.00	10.10	11.40
Qwen3.5-2B	**100.00**	**100.00**	68.00	**100.00**	92.00	7.70	4.80
Qwen3.5-9B	**100.00**	**100.00**	**88.00**	33.33	**95.00**	**1.85**	**2.47**

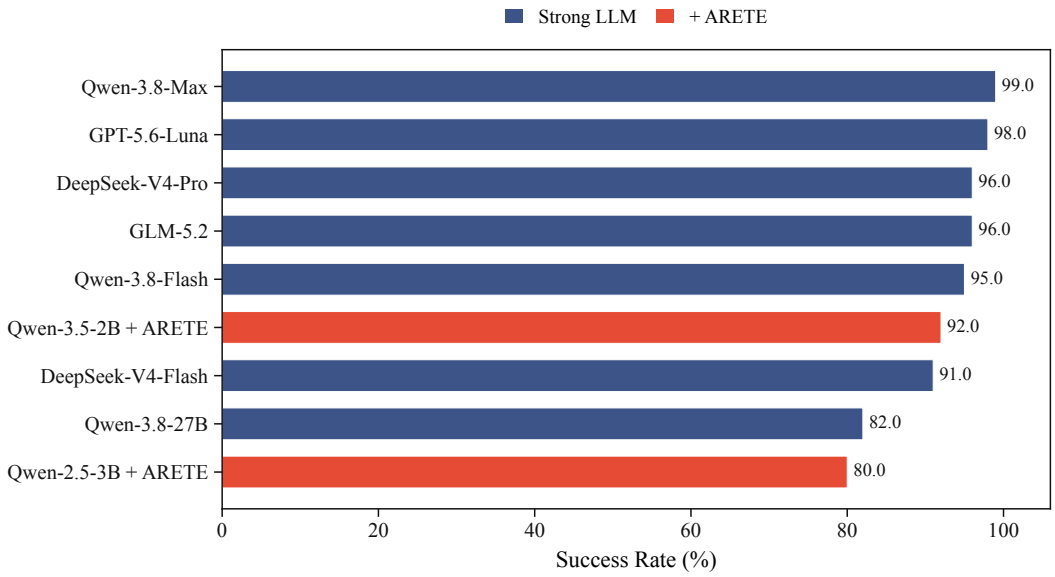


Figure 5: TextCraft success rates of ARETE-trained compact agents and larger general-purpose LLMs. Larger models use ReAct, while compact backbones use ARETE.

Comparison with larger models. Figure 5 contextualizes ARETE-trained compact agents against larger general-purpose models. Qwen3.5-2B with ARETE reaches 92.0% success, exceeding DeepSeek-V4-Flash and Qwen3.8-27B, while remaining below the strongest evaluated models. Qwen2.5-3B reaches 80.0%, close to Qwen3.8-27B at 82.0%. The additional Qwen3.5-9B result in Table 9 reaches 95.0%, matching Qwen3.8-Flash. These comparisons show that targeted training can narrow the task-performance gap between compact agents and substantially larger models; they do not directly measure inference cost or deployment efficiency.

Training dynamics. Figure 6 shows the training-batch success rates of ARETE, ScalingInter-RL, and GRPO on TextCraft. ARETE improves more rapidly during the early and middle stages of training, reaches a high-success regime earlier, and maintains higher success rates than both baselines over most optimization steps. GRPO makes relatively rapid initial progress but subsequently plateaus at a lower level, whereas ScalingInter-RL improves more gradually and narrows the gap only in the later stage of training. Despite local fluctuations, ARETE sustains a stronger training trajectory and ends training at a higher observed success level.

This pattern is consistent with the design of ARETE, which jointly regulates reasoning within tasks and practice across tasks. State-dependent reasoning regulation provides targeted feedback for mode selection and execution, while capability-adaptive practice concentrates training on tasks that remain informative for the current policy. Together, these mechanisms provide more targeted learning opportunities as training progresses, which is reflected in the faster and more sustained improvement of training success. As success rates are measured on the training batches sampled by each method, we use these curves to analyze training efficiency, while relying on the fixed held-out test set for cross-method evaluation.

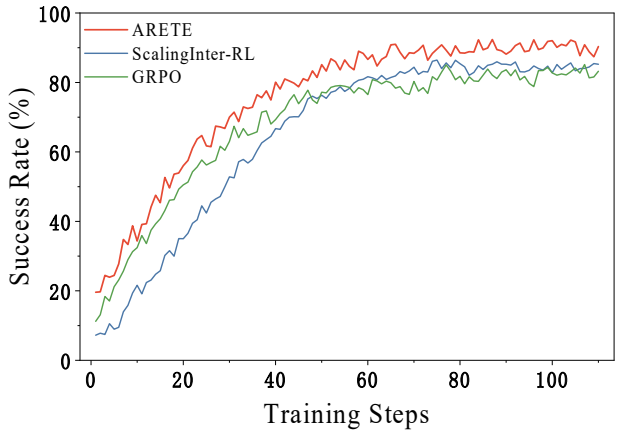


Figure 6: Training dynamics on TextCraft. ARETE reaches a high-success regime earlier and maintains higher training-batch success rates than GRPO and ScalingInter-RL over most optimization steps.

F Qualitative Case Studies

This appendix presents two qualitative case studies from TextCraft and ScienceWorld. Each case contrasts the interaction behavior of ScalingInter-RL and ARETE on a long-horizon task, with particular attention to how the agents respond to execution feedback and revise their subsequent actions.

F.1 Case Study: TextCraft

We first examine a Depth3 TextCraft task, `textcraft_450`, whose goal is to craft cyan terracotta. As shown in Figure 7, the two trajectories differ substantially in how they respond when the required prerequisites for the target craft are not yet satisfied. ScalingInter-RL does not complete the task within 30 recorded actions, whereas ARETE completes it in seven actions with an episode reward of 1.0.

F.1.1 Task Setup

The target item is cyan terracotta. Its relevant dependency chain requires eight terracotta and one cyan dye; cyan dye is synthesized from blue dye and green dye, while blue dye can be produced from either lapis lazuli or cornflower. Figure 7 shows selected steps from both trajectories together with the goal-relevant recipes. Because several intermediate steps in the ScalingInter-RL trajectory are omitted for readability, the figure focuses on the decisions most relevant to the observed failure and recovery patterns.

F.1.2 ScalingInter-RL

ScalingInter-RL initially makes progress by obtaining lapis lazuli and crafting blue dye. It later obtains the required terracotta, but its subsequent target-level crafting attempts repeatedly fail because the prerequisites for cyan terracotta have not been successfully resolved. At Step 6, the agent attempts to craft cyan terracotta and receives the feedback `Could not find enough items to craft minecraft:cyan terracotta`.

The same pattern reappears later in the trajectory. At Step 8, the agent crafts blue dye again, yet at Step 9 its reasoning states that cyan dye is available and it immediately retries the target craft. The environment again rejects the attempt for insufficient items. A similar target-level retry is still present at Step 29. By Step 30, the agent has not recovered the missing dependency and instead returns to obtaining terracotta. The episode therefore terminates without completing the target after 30 recorded actions, with an episode reward of 0.0.

TextCraft: Depth 3

Task: `craft cyan terracotta.`
Relevant recipes (excerpt):

```
craft 8 cyan terracotta using 8 terracotta, 1 cyan dye
craft 2 cyan dye using 1 blue dye, 1 green dye
craft 1 blue dye using 1 lapis lazuli
craft 1 blue dye using 1 cornflower
```

ScalingInter-RL

Step 1

Thought: To craft cyan terracotta, I need cyan dye and terracotta. I need to first craft cyan dye.
Act: `get 1 lapis lazuli`
Obs: `Got 1 lapis lazuli`

Step 2

Act: `craft 1 blue dye using 1 lapis lazuli`
Obs: `Crafted 1 minecraft:blue_dye`

`... Steps 3--4 omitted ...`

Step 5

Thought: It seems I don't have enough terracotta to craft cyan terracotta. I need to find or craft more terracotta first.
Act: `get 8 terracotta`
Obs: `Got 8 terracotta`

Step 6

Thought: Now I have enough terracotta, I can craft cyan terracotta.
Act: `craft 8 cyan terracotta using 8 terracotta, 1 cyan dye` ✗
Obs: `Could not find enough items to craft minecraft:cyan_terracotta`

`... Step 7 omitted ...`

Step 8

Act: `craft 1 blue dye using 1 lapis lazuli`
Obs: `Crafted 1 minecraft:blue_dye`

Step 9

Thought: Now I have cyan dye, I can proceed to craft cyan terracotta.
Act: `craft 8 cyan terracotta using 8 terracotta, 1 cyan dye` ✗
Obs: `Could not find enough items to craft minecraft:cyan_terracotta`

`... Steps 10--28 omitted ...`

Step 29

Thought: Now I have cyan dye, I can proceed to craft cyan terracotta.
Act: `craft 8 cyan terracotta using 8 terracotta, 1 cyan dye` ✗
Obs: `Could not find enough items to craft minecraft:cyan_terracotta`

Step 30

Thought: It seems I still don't have enough terracotta to craft cyan terracotta. I need to find or craft more terracotta first.
Act: `get 8 terracotta`
Obs: `Got 8 terracotta`

Outcome: not completed after 30 recorded actions.

Episode reward: 0.0

ARETE

Step 1 `[Forethought_mode]`

Reasoning: To craft cyan terracotta, need cyan dye and terracotta. Get terracotta first.
Act: `get 8 terracotta`
Obs: `Got 8 terracotta`

Step 2 `[Performance_mode]`

Reasoning: To craft cyan terracotta, need cyan dye. Obtain blueprint to make it.
Act: `get 1 blue dye`
Obs: `Could not find blue dye`

Step 3 `[SelfReflection_mode]`

Reasoning: Failed because we lack lapis lazuli or cornflower; get one of those to craft the missing blue dye.
Act: `get 1 lapis lazuli` ✓
Obs: `Got 1 lapis lazuli`

Step 4 `[Performance_mode]`

Reasoning: With lapis lazuli, can immediately craft blue dye.
Act: `craft 1 blue dye using 1 lapis lazuli`
Obs: `Crafted 1 minecraft:blue_dye`

Step 5 `[Performance_mode]`

Reasoning: Now have blue dye, need cyan dye to craft cyan terracotta.
Act: `get 1 green dye`
Obs: `Got 1 green dye`

Step 6 `[Performance_mode]`

Reasoning: Both dye (blue and green) are in inventory; craft cyan dye.
Act: `craft 2 cyan dye using 1 blue dye, 1 green dye` ✓
Obs: `Crafted 2 minecraft:cyan_dye`

Step 7 `[Performance_mode]`

Reasoning: Have cyan dye and terracotta, now craft cyan terracotta.
Act: `craft 8 cyan terracotta using 8 terracotta, 1 cyan dye` ✓
Obs: `Crafted 8 minecraft:cyan_terracotta`

Outcome: completed in 7 actions.

Episode reward: 1.0

Figure 7: Selected training rollouts on `textcraft_450` (Depth 3). The left trajectory repeatedly attempts the target recipe without producing the required cyan dye and remains unsuccessful after 30 recorded actions. On the right, the agent recovers from a failed direct acquisition through `SelfReflection_mode`, obtains the required upstream ingredient, and resumes `Performance_mode` to synthesize cyan dye and complete the target. Red × symbols mark key failure points in the left trajectory, while green check marks highlight the corresponding recovery and correct-execution behaviors on the right.

The displayed behavior illustrates a failure to consistently connect environment feedback with the unresolved prerequisite structure: although the target-level action repeatedly fails, the subsequent decisions do not reliably establish the missing cyan-dye dependency.

F.1.3 ARETE

ARETE follows a different interaction pattern. At Step 1, `Forethought_mode` identifies the two top-level requirements, cyan dye and terracotta, and begins by obtaining the required terracotta. At Step 2, the agent uses `Performance_mode` and attempts to obtain blue dye directly, but the environment responds that blue dye cannot be found.

The failed acquisition is followed by a change in both reasoning mode and action. At Step 3, the agent switches to `SelfReflection_mode` and adopts a recipe-based alternative, obtaining lapis lazuli to produce blue dye. Once this prerequisite is available, the corrective route becomes clear. The agent returns to `Performance_mode` at Step 4, crafts blue dye, obtains green dye at Step 5, and combines the two dyes to produce cyan dye at Step 6. It then crafts cyan terracotta at Step 7 and completes the task with an episode reward of 1.0.

Importantly, self-reflection is localized to the point at which the previous acquisition strategy fails. After an alternative route has been established, the agent returns to performance-oriented execution rather than continuing to reflect during the remaining routine steps.

F.1.4 Case Analysis

The central contrast in Figure 7 is not merely whether an individual action fails, but how subsequent decisions use the resulting feedback. ScalingInter-RL repeatedly returns to target-level crafting without reliably resolving the missing prerequisite and remains unsuccessful after 30 actions. In contrast, ARETE responds to the failed direct acquisition by moving from `Performance_mode` to `SelfReflection_mode`, selecting an upstream ingredient, and then returning to `Performance_mode` once the corrective route is established.

This trajectory provides a concrete example of a feedback-driven mode transition: execution feedback prompts changes in both reasoning mode and action strategy, while reflection terminates once the next effective action becomes clear. The example serves as a qualitative illustration of this interaction mechanism rather than a causal or aggregate comparison based on a single task.

F.2 Case Study: ScienceWorld

Figure 8 illustrates contrasting responses to an unresolved conductivity test in ScienceWorld. The example focuses on whether an agent revisits the test setup when its current observations are insufficient to determine the target property.

F.2.1 Task and Shared Setup

We next consider an electrical-conductivity task in ScienceWorld. Figure 8 contrasts two continuations from the same partially assembled test circuit. The example examines how the agents respond when the initial observation of an unlit bulb is insufficient to determine whether the tested substance is non-conductive or the circuit itself is incomplete.

F.2.2 ScalingInter-RL

ScalingInter-RL initially assumes that the test circuit is ready and checks the bulb. After observing that the bulb is off, it waits and then checks the bulb again. The same inspection–waiting pattern is repeated in the displayed trajectory: the agent alternates between `look at red light bulb` and `wait`, while the bulb continues to remain off.

Crucially, the agent does not revisit the circuit configuration or inspect the electrical connections. As a result, the underlying missing connection remains unresolved. The same behavior persists through the continuation, and the trajectory reaches its 30-action budget without completing the task. Steps 7–28 are omitted from the figure for readability.

This behavior should be interpreted as persistence in an unproductive testing strategy rather than as repeated invalid commands: the displayed actions are executable, but they do not produce the diagnostic information required to resolve the task.

F.2.3 ARETE

ARETE begins in `Forethought_mode` by establishing the high-level order of testing the substance and then classifying it from the observed result. It checks the bulb at Step 1 and uses `Performance_mode` to wait for one interval at Step 2. The available feedback does not yet resolve whether the substance is conductive, so the agent changes its diagnostic strategy at Step 3.

In `SelfReflection_mode`, the agent considers an incomplete circuit as a possible explanation for the unlit bulb and chooses to inspect the battery terminals. This diagnostic action provides new evidence: the battery anode is connected to the red wire, whereas the cathode is connected to nothing.

Once the missing return connection is directly observed, the next corrective action becomes clear. The agent returns to `Performance_mode` at Step 4 and connects the free end of the green wire to the battery cathode. It then checks the bulb again at Step 5 and observes that the bulb is on. Only after verifying the repaired circuit does it classify the substance as conductive and move it to the blue box at Step 6. The continuation therefore completes in six actions after the shared setup.

F.2.4 Case Analysis

Figure 8 highlights a different form of feedback-responsive regulation from the TextCraft example. Here, the key problem is not an explicitly rejected command, but an unresolved observation that does not yet distinguish between a non-conductive substance and an incomplete test circuit.

ScalingInter-RL continues to inspect the same outcome without changing how information is gathered. ARETE instead switches from `Performance_mode` to `SelfReflection_mode`, selects a diagnostic action that exposes the missing connection, and then returns to `Performance_mode` to carry out the repair and complete the classification.

Together with the TextCraft case, this example shows that self-reflection is followed by a change in the agent's subsequent decision rather than extended verbal deliberation. In TextCraft, reflection after a failed acquisition redirects the agent toward an upstream prerequisite, whereas in ScienceWorld, it shifts the agent toward diagnostic inspection when the current testing strategy remains inconclusive. Together, these cases qualitatively illustrate how reasoning modes adapt to interaction feedback, without implying a causal effect of explicit mode selection from individual trajectories alone.

ScienceWorld: Electrical Conductivity

Task: Determine whether `unknown substance B` is electrically conductive. First, focus on the substance. If conductive, place it in the `blue box`; otherwise, place it in the `orange box`.

Shared setup: Both agents have focused on the substance and inserted it into a test circuit. The `red light bulb` is off. Each agent has a budget of 30 subsequent actions.

Available equipment: A battery, a red wire, a green wire, and a red light bulb.

ScalingInter-RL

Step 1

Thought: The test circuit is ready. Check whether the bulb lights.
Act: `look at red light bulb`
Obs: `The red light bulb is off.`

Step 2

Thought: The bulb may need more time to respond.
Act: `wait`
Obs: `Time passes.`

Step 3

Thought: Check the bulb again.
Act: `look at red light bulb`
Obs: `The red light bulb is still off.` ✗

Step 4

Thought: Wait longer before checking again.
Act: `wait`
Obs: `Time passes.`

Step 5

Thought: Enough time may have passed. Check whether the bulb has lit.
Act: `look at red light bulb`
Obs: `The red light bulb is still off.` ✗

Step 6

Thought: Allow more time for the test.
Act: `wait`
Obs: `Time passes.`

`... Steps 7--28 omitted ...`

Step 29

Thought: The bulb may have responded by now. Check it once more.
Act: `look at red light bulb`
Obs: `The red light bulb is still off.` ✗

Step 30

Thought: The bulb has not lit yet. Allow more time for the test.
Act: `wait`
Obs: `Time passes.`

Outcome: Not completed within the 30-action budget. The circuit remains incomplete and conductivity unresolved.

ARETE

Step 1 `[Forethought_mode]`

Reasoning: Test the substance in the circuit, then classify it from the observed result.
Act: `look at red light bulb`
Obs: `The red light bulb is off.`

Step 2 `[Performance_mode]`

Reasoning: Allow one interval before checking the circuit.
Act: `wait`
Obs: `Time passes.`

Step 3 `[SelfReflection_mode]`

Reasoning: Waiting alone does not verify the circuit. An unlit bulb may indicate a missing connection. Inspect the battery terminals. ✓
Act: `look at battery`
Obs: `The anode is connected to the red wire. The cathode is connected to nothing.`

Step 4 `[Performance_mode]`

Reasoning: The return connection is missing. Connect the free end of the green wire to the battery cathode.
Act: `connect green wire terminal 2 to battery cathode` ✓
Obs: `The green wire is connected to the battery cathode.`

Step 5 `[Performance_mode]`

Reasoning: The circuit is now closed. Check the bulb before classifying the substance.
Act: `look at red light bulb`
Obs: `The red light bulb is on.`

Step 6 `[Performance_mode]`

Reasoning: The bulb lights with the substance in the test path, supporting the conductive classification.
Act: `move unknown substance B to blue box` ✓
Obs: `The substance is placed in the blue box.`

Outcome: Completed in 6 actions after the shared setup. The circuit is repaired and the substance correctly classified.

Figure 8: Comparison of ScalingInter-RL and ARETE trajectories on a ScienceWorld electrical-conductivity task. Both agents start from the same incomplete test circuit and have a budget of 30 additional actions. ScalingInter-RL (left) repeatedly alternates between inspecting the bulb and waiting, leaving the circuit unrepaired and exhausting its action budget without completing the task; steps 7–28 are omitted for readability. ARETE (right) switches from `Performance_mode` to `SelfReflection_mode`, inspects the battery terminals, and identifies the missing return connection. It then returns to `Performance_mode` to repair the circuit, verify that the bulb lights, and correctly classify the substance, completing the task in six actions. Red × symbols mark repeated checks without repair, while green check marks highlight diagnosis, repair, and classification. All step numbers refer to actions taken after the shared setup.

G DISCUSSION

Long-horizon agents may invoke a language model at every interaction step. As interaction histories and environment observations accumulate, repeatedly processing increasingly long contexts raises token consumption, latency, and inference cost. When these agents rely on large models accessed through online services, prompts, interaction histories, environment observations, and tool outputs may also need to be transmitted to external servers. This creates practical data-governance concerns in domains such as education, healthcare, finance, and enterprise services, where interactions may contain personal or proprietary information. ARETE targets this setting by strengthening compact 2B-3B policies through adaptive training rather than increasing backbone size. Scaffolded task construction, comparisons between unaided and assisted performance, and practice allocation are used only during training; the resulting policy executes tasks independently at inference. This separation supports local deployment and reduces reliance on repeated calls to large external models.

ARETE is not intended to claim that compact agents universally replace large general-purpose models, which may retain advantages in broad knowledge and open-domain generalization. Its practical value instead lies in recurring, domain-focused, long-horizon workloads, where the same trained policy can be reused across many interactions and where inference cost and data control are central concerns. Although adaptive training introduces an upfront cost for training and task verification, this cost can be amortized across repeated deployment, while no task generator or scaffolding mechanism is required online. The competitive performance of ARETE's compact backbones relative to substantially larger models suggests that adaptive regulation can improve the tradeoff between agent capability and deployment cost without expanding model size. Future work will evaluate longer and more complex task sequences, quantify end-to-end token use, latency, and cost on realistic hardware, and develop domain-specific verification and deployment mechanisms for privacy-sensitive applications.