3SPO: STATE-SCORE-SUPERVISED POLICY OPTIMIZATION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training large language models (LLMs) as autonomous agents via reinforcement learning (RL) has enabled frontier models to achieve superhuman performance in long-horizon tasks. However, existing RL algorithms operate at the trajectory level, performing policy optimization only after collecting complete episode rollouts. This coarse-grained approach faces fundamental challenges in multi-turn agent settings where rewards are sparse, delayed, and credit assignment across individual steps is critical. In this work, we propose **State-Score-Supervised Policy Optimization (3SPO)**, a novel RL algorithm that performs post-step policy optimization with dynamic state score supervision. At each step, 3SPO computes the state score based on historical success rates, supervising step-wise credit assignment, adaptive rollout and post-step policy optimization without requiring value function estimation or additional auxiliary models. Theoretically, under a per-state bandit abstraction, we show that the proposed score-supervised allocation mechanism achieves logarithmic allocation regret and provide sample-complexity guarantees for action identification, score distinguishability, and filtering stability. Experiments on ALFWorld and WebShop with Qwen2.5-1.5B/7B-Instruct show that 3SPO consistently outperforms GRPO by $+22.6\%$ on ALFWorld and $+14.5$ points on WebShop, while using comparable resources to achieve $2.4\times$ more state exploration and $1.8\times$ faster convergence.

1 INTRODUCTION

Large language models (LLMs) (Achiam et al., 2023; Team et al., 2023; Yang et al., 2024; Guo et al., 2025) have demonstrated remarkable capabilities as autonomous agents across diverse interactive environments. In embodied task planning (Shridhar et al., 2020), agents navigate household spaces and complete multi-step goals; in web interaction (Yao et al., 2022a), they browse, filter, and purchase items matching complex queries; in code generation (Roziere et al., 2023), they synthesize programs from natural-language specifications. In these multi-turn settings, agents must perceive states, reason over history, and perform long-horizon planning and decision.

Group-based reinforcement learning has emerged as a primary approach for training LLM agents through environment interaction. Critic-free methods such as GRPO (Shao et al., 2024), RLOO (Ahmadian et al., 2024), and their variants (Hu, 2025; Yu et al., 2025) compute relative advantages within groups of rollouts, avoiding the memory overhead and estimation errors of learned value functions. However, when applied to long-horizon agent tasks, these methods typically operate at the trajectory level: a single episode-level return is assigned to all actions in a rollout, regardless of their individual contribution. This uniform advantage signal obscures fine-grained distinctions across episodes spanning dozens of steps, while sparse terminal rewards provide little guidance on whether intermediate decisions make progress or lead the agent into failure modes. Recent attempts address this issue through experience replay (Lu et al., 2025a), semi-online RL (Lu et al., 2025b), self-refinement loops (Madaan et al., 2023), or step-wise credit assignment methods such as GiGPO (Feng et al., 2025) and HGPO (Jiang et al., 2025). While these approaches recover more informative training signals than trajectory-level baselines, they either require domain-specific engineering or still defer policy updates until complete trajectories are collected. As a result, incorrect intermediate decisions may propagate through subsequent steps before the policy receives any corrective signal, making optimization inefficient and unstable in multi-turn agent environments.

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.

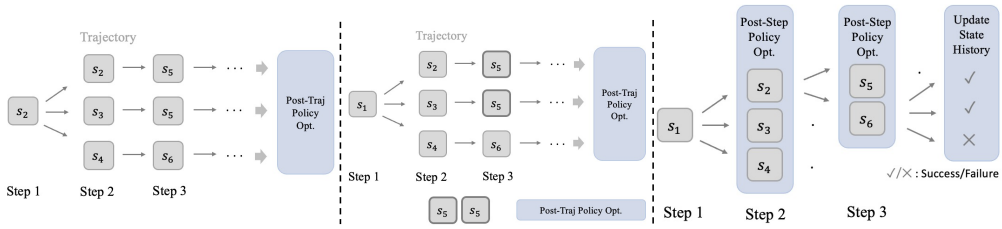


Figure 1: Comparison of multi-turn LLM agent training. **Left**: Rollout of GRPO is a group of trajectories. **Middle**: GiGPO introduces state-level grouping but still performs post-trajectory policy optimization. **Right**: 3SPO computes a continuous state score from historical interaction statistics that simultaneously supervises step-wise credit assignment and adaptive rollout allocation, enabling post-step policy optimization.

A key observation is that, in multi-turn agent tasks, environment states are not merely transient contexts but reusable decision points whose historical outcomes reveal how well the current policy has mastered the task. States that frequently lead to successful completion are likely mastered, whereas states with low but non-zero success probability often indicate learnable bottlenecks where exploration are most needed. This motivates using state interaction statistics as an online proxy for task completion status and learning potential, enabling step-wise credit assignment and training.

To instantiate this idea, we propose **State-Score-Supervised Policy Optimization (3SPO)**, a novel RL algorithm that performs step-wise credit assignment, adaptive rollout and post-step policy optimization with dynamic state-score-supervision. The state score measures how unresolved a state remains under the current policy: states with high historical success rates receive lower scores because they are likely mastered, whereas states with low but still recoverable success rates receive higher scores because they expose difficult yet learnable bottlenecks. In this sense, $S(s_t)$ converts sparse trajectory-level outcomes into an online step-level training signal. As illustrated in Fig. 1, 3SPO simultaneously: (i) governs adaptive rollout allocation, concentrating computation on unresolved states; and (ii) enables post-step policy optimization with a composite reward that balances exploration, credit assignment, and task performance. Both components arise naturally from different uses of the state score. Specifically, the score difference $S(s_t) - S(s_{t+1})$ provides transition-level credit, the magnitude of $S(s_t)$ determines sampling priority, and the online availability of the score enables policy updates immediately after each step rather than after trajectory completion.

We provide both theoretical and empirical validation of 3SPO. Theoretically, we analyze 3SPO under a per-state bandit abstraction and establish convergence guarantees for the state score mechanism. Empirically, we evaluate 3SPO on ALFWorld and WebShop with Qwen2.5-1.5B/7B-Instruct. Our contributions are as follows:

- We introduce a dynamic state score derived from historical interaction statistics, which measures task computation status and learning potential and provides an online step-level signal for agent training. We also provide theoretical guarantees for the state score mechanism.
- We propose **3SPO**, a step-level RL algorithm that uses the state score to supervise reward construction, adaptive rollout allocation, and post-step policy optimization for LLM agents.
- Experiments on ALFWorld and WebShop show that 3SPO consistently outperforms strong prompting and RL baselines, achieving state-of-the-art results on the evaluated benchmarks.

2 Related Work

LLMs as Agents. LLMs have been deployed as agents across diverse interactive environments. Prompting-based methods such as ReAct (Yao et al., 2022b) and Reflexion (Shinn et al., 2023) operate with frozen parameters, interleaving reasoning traces with action execution and using self-generated feedback for iterative improvement. Tool-augmented LLMs (Qin et al., 2023; Shen et al., 2023) connect to external APIs and services, enabling information retrieval and service invocation, while memory-augmented architectures (Packer et al., 2023; Park et al., 2023) address long-term context management through hierarchical memory and social simulation. Despite their breadth of

applications, these methods share a fundamental limitation: without parameter updates, the policy cannot adapt its behavior based on accumulated experience, making them brittle in environments requiring systematic exploration or recovery from failure modes not covered by pretraining.

Reinforcement Learning for LLMs. PPO (Schulman et al., 2017) has served as the backbone of RLHF pipelines (Ouyang et al., 2022; Bai et al., 2022), building on trust region theory (Schulman et al., 2015a; Kakade, 2001) and GAE (Schulman et al., 2015b), though its learned value function increases memory overhead and introduces estimation errors. Critic-free group-based methods such as GRPO (Shao et al., 2024), RLOO (Ahmadian et al., 2024; Kool et al., 2019), REINFORCE++ (Hu, 2025), and DAPO (Yu et al., 2025) compute relative advantages within rollout groups, eliminating the need for a value network while maintaining training stability. These methods have been primarily demonstrated on single-turn tasks such as mathematical reasoning (Cobbe et al., 2021; Hendrycks et al., 2021; Guo et al., 2025) and code generation (Roziere et al., 2023). However, extending group-based RL to multi-turn agent settings requires per-step credit assignment, as trajectory-level advantages collapse fine-grained distinctions across long decision sequences.

Step-Wise Credit Assignment for LLM Agents. Credit assignment in long-horizon tasks with sparse rewards has been addressed through temporal difference learning (Sutton, 1988; Sutton & Barto, 1998), GAE (Schulman et al., 2015b), count-based exploration (Bellemare et al., 2016; Ostrovski et al., 2017), curiosity-driven methods (Burda et al., 2018; Pathak et al., 2017; Fu et al., 2017), counterfactual baselines (Foerster et al., 2018; Lowe et al., 2017) and value decomposition (Rashid et al., 2020; Sunehag et al., 2017). In LLM agent settings, STEP (Chen et al., 2025) maintains a smoothed success-rate record per state to guide adaptive trajectory resampling and decompose trajectories into step-level training samples. GiGPO (Feng et al., 2025) introduces anchor state grouping for step-level advantage computation, retroactively clustering identical states across completed trajectories. Hierarchical methods (He et al., 2026; Kulkarni et al., 2016) structure credit across temporal abstraction levels, while process-supervised rewards (Lightman et al., 2023; Zheng et al., 2025) provide step-level feedback at the cost of additional training data and model components. 3SPO's key innovation lies in post-step policy updates rather than deferred trajectory-level optimization.

3 METHOD

In this section, we first formulate the multi-turn LLM agent training problem (Sec. 3.1) and then introduce 3SPO, a state-score-supervised policy optimization framework for long-horizon agentic tasks. As illustrated in Fig. 2, 3SPO maintains a dynamic state score computed from historical interaction statistics and uses it as a unified supervisory signal throughout training. The framework consists of three key components: Dynamic State Score Function (Sec. 3.2) quantifies the difficulty and learning potential of each state; Step-Wise Reward Model (Sec. 3.3) converts sparse trajectory-level outcomes into fine-grained transition-level rewards; Adaptive Rollout and Post-Step Policy Optimization (Sec. 3.4) allocates more rollouts to informative states and updates the policy immediately after each step. Finally, we provide theoretical insights into the state-score-supervised allocation mechanism in Sec. 3.5.

3.1 PROBLEM FORMULATION

The state $s \in \mathcal{S}$ comprises the current environment observation. At each step t, the agent π_θ parameterized by θ observes the state s_t and generates a textual action $a_t \in \mathcal{V}^n$, where $\mathcal{V}$ denotes the token vocabulary and n is the maximum generation length. Upon execution, the environment returns a scalar reward $r_t \in \mathcal{R}$ and the next state s_{t+1}. This process repeats until the task is completed or a terminal state is reached, yielding a trajectory $\tau^T = \{(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_T, a_T, r_T)\}$.

3.2 DYNAMIC STATE SCORE FUNCTION

To quantify the difficulty and learning potential of each interaction state s_t in real time, we propose a dynamic state-score function $S(s_t)$, which is updated smoothly based on the agent's historical interaction statistics on s_t:

$$S(s_t) = \exp\left(-\lambda(t) \cdot \frac{N_{\text{success}}(s_t)}{N_{\text{total}}(s_t) + \epsilon}\right) \cdot \mathbf{1}\left\{N_{\text{fail}}(s_t) < \xi \vee \frac{N_{\text{success}}(s_t)}{N_{\text{total}}(s_t) + \epsilon} > \zeta\right\}, \tag{1}$$

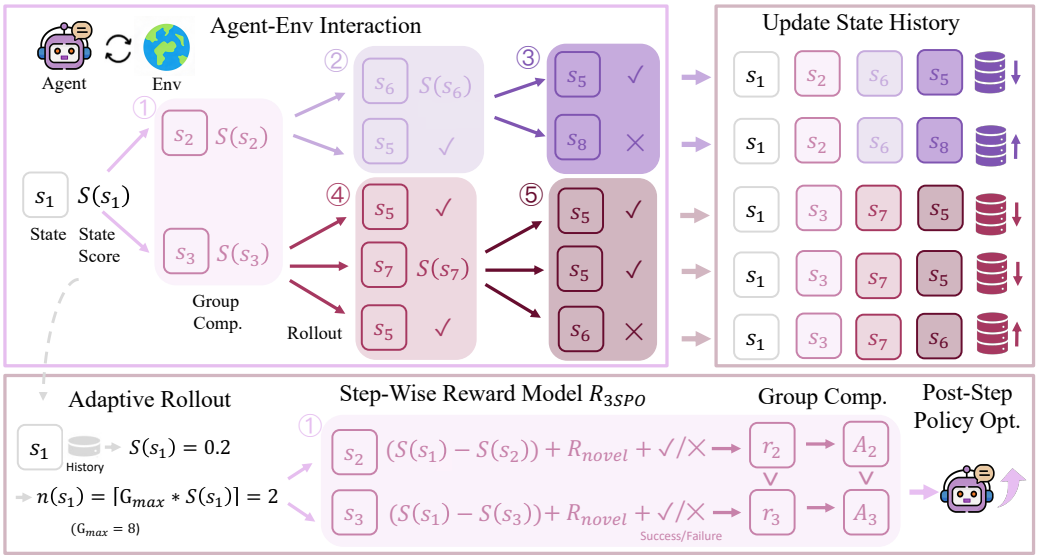


Figure 2: Overview of 3SPO. The agent interacts with the environment to produce state transitions. A dynamic state score $S(s_t)$ computed from historical visit statistics simultaneously supervises: (i) step-wise reward R_{3SPO}, (ii) adaptive rollout allocation $n(s_t)$, and (iii) post-step policy optimization. State history is updated at the end of each trajectory, forming a closed loop learning pipeline.

where $N_{\text{total}}(s_t)$ denotes the total number of completed trajectories passing state s_t, $N_{\text{success}}(s_t)$ is the number of successful outcomes from s_t and $N_{\text{fail}}(s_t) = N_{\text{total}}(s_t) - N_{\text{success}}(s_t)$ is the number of failures. $\lambda(t) = \alpha \log t$ is a time-varying annealing coefficient that balances exploration at early step and exploitation at later step, $\xi, \zeta > 0$ are threshold hyperparameters that define the valid range of states, and ϵ is a small constant to avoid division by zero. The state-score function exhibits two key properties: (i) *difficulty-aware learning potential quantification*. The exponential term assigns higher scores to states with lower historical success rates, prioritizing learning on challenging states; (ii) *exploration truncation*. States too difficult to learn are assigned zero score, preventing the algorithm from wasting resources on intractable states that are unlikely to yield meaningful progress.

3.3 Step-Wise Reward Model

To resolve the sparse credit assignment problem in long-horizon agent tasks, we propose a step-wise reward model $R_{\text{3SPO}}(s_t, s_{t+1})$, which integrates three complementary reward components:

$$\begin{aligned} R_{\text{3SPO}}(s_t, s_{t+1}) = {} & \omega(N_{\text{total}}(s_t)) \cdot R_{\text{novel}}(s_{t+1}) \\ & + (0.5 - \omega(N_{\text{total}}(s_t))) \cdot (S(s_t) - S(s_{t+1})) \\ & + 0.5 \cdot R_{\text{success}}(s_{t+1}). \end{aligned} \tag{2}$$

Specifically: (i) *novelty reward* $R_{\text{novel}}(s_{t+1}) = \mathbf{1}\{s_{t+1} \notin \tau^t\}$ encourages the agent to perform actions that induce observable state changes rather than repetitive or ineffective operations that leave the interface unchanged; (ii) *state score difference reward* $S(s_t) - S(s_{t+1})$ provides credit assignment by rewarding transitions from high-difficulty to low-difficulty states; and (iii) *terminal task reward* $R_{\text{success}}(s_{t+1})$, a score signal indicating task completion. These components remain dynamically balanced via the weight function $\omega(N_{\text{total}}(s_t)) = 0.5 \cdot \exp(-\gamma N_{\text{total}}(s_t))$, which decays exponentially with state visit count (the choice of γ is detailed in Sec. 3.5). It prioritizes the novelty reward for rarely visited states to encourage exploration, while emphasizes the state score difference reward for frequently visited states to facilitate the exploitation of learned knowledge.

3.4 Adaptive Rollout and Post-Step Policy Optimization

We propose an adaptive rollout rule that dynamically adjusts the number of rollout based on state score to address the inefficiency of uniform sampling. The number of rollout $n(s_t)$ for state s_t is defined as:

$$n(s_t) = \lceil G_{\text{max}} \cdot S(s_t) \rceil, \tag{3}$$

where G_{max} is the maximum number of rollout ($n(s_t) = 1$ proceeds to the next state without policy optimization, and $n(s_t) = 0$ truncates the trajectory immediately). This rollout rule enables adaptive resource allocation: states with higher scores receive more rollout, maximizing the informativeness of each sampling step while reducing waste on mastered states.

Leveraging the adaptive rollout mechanism together with the state score function and the reward model, we design a post-step policy optimization pipeline without waiting for full trajectory completion. The core optimization objective is:

$$\mathcal{J}_{\text{3SPO}}(\theta) = \mathbb{E}_{\substack{s_t \sim \mathcal{S} \\ \{a^i\}_{i=1}^{n(s_t)} \sim \pi_{\theta_{\text{old}}}}} \left\{ \frac{1}{n(s_t)} \sum_{i=1}^{n(s_t)} \frac{1}{|a^i|} \sum_{k=1}^{|a^i|} \min \left[r_{i,k}(\theta) \hat{A}_{i,k}, \text{clip}\big(r_{i,k}(\theta), 1-\epsilon, 1+\epsilon\big) \hat{A}_{i,k} \right] - \beta \mathbb{D}_{KL} \left[\pi_\theta \parallel \pi_{\text{ref}} \right] \right\}, \tag{4}$$

where $r_{i,k}(\theta) = \frac{\pi_\theta(a^{i,k}|a^{i,<k}, s_t)}{\pi_{\theta_{old}}(a^{i,k}|a^{i,<k}, s_t)}$ is the policy ratio and $\hat{A}_i = \frac{R_{\text{3SPO}i} - \text{mean}(\{R_{\text{3SPO}i}\})}{\text{std}(\{R_{\text{3SPO}i}\})}$ is the group-normalized relative advantage computed using R_{3SPO}. Unlike trajectory-level GRPO variants that aggregate rewards across entire trajectories before updating, 3SPO performs policy optimization immediately after collecting rollouts from the current state s_t. Instead of committing to a single greedy trajectory, 3SPO employs ranked backtracking DFS. This mechanism not only accelerates state score convergence in the early stages by prioritizing high reward branches but also allows the agent to systematically explore alternative paths whenever a path reaches a termination. The complete pipeline is summarized in Algorithm 1 (Appendix A.1).

3.5 Theoretical Insights

We analyze 3SPO under a finite-arm bandit abstraction at each decision state. Each action a at state s_t corresponds to an arm with unknown success probability $p^*(a) \in [0, 1]$, representing the probability of eventual task completion. Given the one-to-one correspondence between each action a and its subsequent state s_{t+1}, the state score $S(a) \triangleq S(s_{t+1}) = \exp(-\lambda(t)\hat{p}(a))$ with $\lambda(t) = \alpha \log t$ assigns higher scores to actions with lower empirical success rates $\hat{p}(a)$, supervising rollout allocation toward unresolved states. Let $\mathcal{A} = \{1, \ldots, A\}$ denote the finite action set, $N(a)$ the cumulative samples of action a, and $\Delta_a = p^*(a) - p^*(a_w)$ the gap relative to the lowest-success action $a_w = \arg\min_a p^*(a)$. Full proofs of all results are deferred to Appendix A.2. Experimental validation is presented in Sec. 4.3.

Concentration of Empirical Estimates. Define the "good event" at iteration i as $\mathcal{E}_i = \{\forall a \in \mathcal{A} : |\hat{p}(a) - p^*(a)| \leq \sqrt{\frac{\log A}{N(a)}}\}$. The empirical success rate concentrates around its true value under $\mathcal{E}_i$ with probability at least $1 - O(i^{-2})$. By the Borel–Cantelli lemma, $\mathcal{E}_i$ holds almost surely for all but finitely many iterations; its complement $\mathcal{E}_i^c$ (the "bad event") has probability decaying as $O(i^{-2})$. This guarantees that historical statistics converge to true success rates .

Score Separation. On the concentration event, for any pair of actions a, b with $\Delta_{a,b} = p^*(b) - p^*(a) > 0$ and $N(a), N(b) \geq T_a$, the score difference satisfies $S(a) - S(b) \geq \lambda(t) \exp(-\lambda(t)) \cdot \frac{\Delta_{a,b}}{2}$, where $T_a = \frac{16 \log i}{\Delta_{a,b}^2}$ is the exploration threshold. This proves that the state score reliably distinguishes actions with different success probabilities.

Non-worst Action Allocation. For any non-worst action $a \neq a_w$ with $N(a), N(a_w) \geq T_a$, the fraction of rollouts allocated satisfies $\frac{n_i(a)}{\sum_{a'} n_i(a')} \leq \exp(-\lambda(t) \cdot \frac{\Delta_a}{2}) = t^{-\frac{\alpha \Delta_a}{2}}$. This ensures that computational resources are preferentially directed toward harder states.

Allocation Regret. We now formally bound the training-time allocation regret of 3SPO's score-supervised rollout allocation. Building on the allocation bound above, the allocation regret over I rounds is:

$$R(I) = \mathbb{E}\left[\sum_{i=1}^{I} \sum_{a \neq a_w} n_i(a) \cdot \Delta_a\right] = O(\log I). \tag{5}$$

This logarithmic bound shows that 3SPO incurs only $O(\log I)$ allocation overhead relative to an oracle unresolved-state sampler under the per-state bandit abstraction. No algorithm can achieve asymptotically smaller regret under the same bandit setting without prior knowledge of the gaps Δ_a. The regret decomposes per action: $R(I) = \sum_{a \neq a_w} \Delta_a \cdot \mathbb{E}[M_I(a)]$, where $M_I(a) = \sum_{i=1}^{I} n_i(a)$. Each $M_I(a)$ splits into exploration ($N(a) < T_a$) and exploitation ($N(a) \geq T_a$) phases. The exploration phase contributes at most $G_{\max} \cdot T_a$ per action. In the exploitation phase, we invoke the non-worst action allocation bound: the rollout fraction is bounded by $\rho_i(a) = \exp(-\alpha \log t \cdot \Delta_a/2)$ under the good event $\mathcal{E}_i$ defined above. With $\alpha \geq \frac{2 \log I}{\Delta_{\min} \log 2}$ and $t \geq 2$, this gives $\rho_i(a) \leq 1/I$. The expectation splits into the good event term ($O(1)$) and the bad event $\mathcal{E}_i^c$ term ($n_i(a) \leq G_{\max}$ with probability $O(i^{-2})$, also $O(1)$).

Weight Schedule. The composite reward uses $\omega(N) = 0.5 \exp(-\gamma N)$ to balance novelty and score-difference rewards. At the exploration threshold $T_a = \frac{16 \log I}{\Delta_a^2}$, setting the two weights equal ($\omega = 0.5 - \omega = 0.25$) yields $\gamma = \frac{\log 2}{T_a} = \frac{\Delta_{\min}^2 \log 2}{16 \log I}$. Before T_a, $\omega(N) > 0.25$ prioritizes novelty; after T_a, the score-difference term dominates.

4 Experiments

4.1 Experiment Setup

Benchmarks. We train LLM agents on ALFWorld (Shridhar et al., 2020) and WebShop (Yao et al., 2022a), which are designed to assess the ability of LLM agents to solve long-history tasks. Benchmark details and supplementary experiments on AppWorld (Trivedi et al., 2024) and a search-augmented QA benchmark are provided in Appendix.

Baselines. We compare against: (1) Closed-source LLMs: GPT-4o (Achiam et al., 2023) and Gemini-2.5-Pro (Team et al., 2023). (2) Prompting agents: ReAct (Yao et al., 2022b) and Reflexion (Shinn et al., 2023). (3) RL methods: PPO (Schulman et al., 2017), RLOO (Kool et al., 2019; Ahmadian et al., 2024), GRPO (Shao et al., 2024), GiGPO (Feng et al., 2025) and HGPO (Jiang et al., 2025).

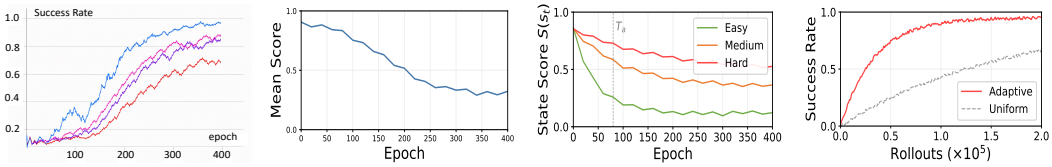
Training Details. We adopt Qwen2.5-1.5B/7B-Instruct (Yang et al., 2024) as our base models. Group-based RL methods use group size $G = 8$. For 3SPO, we set $\alpha = 1$, $\xi = 10$, $\zeta = 0.1$ and $G_{\text{max}} = 8$. The sensitivity analysis of parameters is presented in Appendix A.3.4. The results for Qwen3 and other details are listed in the Appendix.

4.2 Experimental Results

Table 1 shows results across closed models, prompting baselines, and RL training methods. RL training consistently outperforms prompting-only approaches (Qwen2.5, ReAct, Reflexion), with the gap widening on more complex tasks. A pronounced gap exists between trajectory-level methods (PPO, RLOO, GRPO) and step-level methods (GiGPO, HGPO, 3SPO). On both ALFWorld and WebShop, step-level RL-trained methods substantially outperform trajectory-level baselines, confirming that fine-grained credit assignment is crucial for these long-horizon tasks. With Qwen2.5-1.5B-Instruct, 3SPO consistently improves over HGPO by $+2.65\%$ on ALFWorld In-Success and $+3.02\%$ on Out-Success. On WebShop, 3SPO achieves $+4.72\%$ over HGPO in task scores and $+7.92\%$ in task success rates. With Qwen2.5-7B-Instruct, 3SPO remains superior, achieving 96.81% on ALFWorld and 80.45% on WebShop. The gains over HGPO are $+1.37\%/+3.88\%$ on ALFWorld (In/Out-Success) and $+1.94\%$ on WebShop. These suggest that 3SPO's state score supervision scales favorably with model capacity, and enhances model's generalization ability (achieving higher out-success performance compared to other methods). We further observe that 3SPO brings greater gains on the smaller Qwen2.5-1.5B-Instruct model, as its limited agentic capability tends to produce

Table 1: Performance comparison on ALFWorld and WebShop. For ALFWorld, we report the overall success rate (↑) for both *in-distribution* (In-Success) and *out-of-distribution* tasks (Out-Success). For WebShop, we report the average task score (↑) and the average task success rate (↑). Most results are averaged over 3 random seeds during testing. The best results are highlighted in **bold**, and the second-best results are underlined.

Model	Type	Method	ALFWorld		WebShop	
			In-Success	Out-Success	Task Scores	Task Success Rates
Closed	Prompting	GPT-4o	48.0	46.0	31.8	23.7
	Prompting	Gemini-2.5-Pro	60.3	50.5	42.5	35.9
Qwen2.5-1.5B-Instruct	Prompting	Qwen2.5	4.1	–	23.1	5.2
	Prompting	ReAct	12.8	–	40.1	11.3
	Prompting	Reflexion	21.8	–	55.8	21.9
	RL Training	PPO (with critic)	$54.4_{\pm 3.1}$	–	$73.8_{\pm 3.0}$	$51.5_{\pm 2.9}$
	RL Training	RLOO	$69.7_{\pm 2.5}$	$68.7_{\pm 10.7}$	$73.9_{\pm 5.6}$	$52.1_{\pm 6.7}$
	RL Training	GRPO	$72.8_{\pm 3.6}$	$70.1_{\pm 2.5}$	$75.8_{\pm 3.5}$	$56.8_{\pm 3.8}$
	RL Training	GiGPO	$90.16_{\pm 0.22}$	$84.76_{\pm 2.83}$	$84.95_{\pm 2.57}$	$66.53_{\pm 1.92}$
	RL Training	HGPO	$\underline{92.77}_{\pm 1.08}$	$\underline{90.16}_{\pm 0.78}$	$\underline{85.56}_{\pm 2.86}$	$\underline{71.54}_{\pm 4.00}$
	RL Training	**3SPO**	$\mathbf{95.42}_{\pm 0.58}$	$\mathbf{93.18}_{\pm 1.21}$	$\mathbf{90.28}_{\pm 0.53}$	$\mathbf{79.46}_{\pm 1.73}$
Qwen2.5-7B-Instruct	Prompting	Qwen2.5	14.8	–	26.4	7.8
	Prompting	ReAct	31.2	–	46.2	19.5
	Prompting	Reflexion	42.7	–	58.1	28.8
	RL Training	PPO (with critic)	$77.08_{\pm 1.12}$	$76.23_{\pm 1.46}$	$81.4_{\pm 3.1}$	$68.7_{\pm 5.1}$
	RL Training	RLOO	$77.86_{\pm 0.03}$	$73.95_{\pm 0.05}$	$80.3_{\pm 3.2}$	$65.7_{\pm 4.0}$
	RL Training	GRPO	$78.64_{\pm 0.73}$	$76.82_{\pm 1.47}$	$79.3_{\pm 2.8}$	$66.1_{\pm 3.7}$
	RL Training	GiGPO	$93.29_{\pm 0.40}$	$92.18_{\pm 0.39}$	$88.93_{\pm 1.49}$	$77.60_{\pm 1.68}$
	RL Training	HGPO	$\underline{95.44}_{\pm 0.62}$	$\underline{92.05}_{\pm 0.22}$	$\underline{88.96}_{\pm 1.04}$	$\underline{78.51}_{\pm 1.40}$
	RL Training	**3SPO**	$\mathbf{96.81}_{\pm 0.35}$	$\mathbf{95.93}_{\pm 0.67}$	$\mathbf{91.37}_{\pm 0.82}$	$\mathbf{80.45}_{\pm 0.89}$



(a) (b) (c) (d)

Figure 3: Training and state score dynamics on ALFWorld (Qwen2.5-1.5B). (a) 3SPO (Blue) converges faster than GRPO (Red), GiGPO (Purple) and HGPO (Pink). (b) Mean state score decays monotonically as the policy improves. (c) Score distributions shift from high to low as states are mastered across easy/medium/hard tasks. (d) Adaptive rollout converges to $> 90\%$ success within 2.0×10^5 rollouts, while uniform rollout remains at $\sim 66\%$ and still rising.

longer and more redundant rollouts that introduce greater bias in advantage estimation. In this case, the state score mechanism becomes more important for refining and stabilizing the credit signal, rollout allocation, and policy optimization. These results suggest that 3SPO is particularly well-suited to challenging long-horizon agentic tasks where rollouts are lengthy and credit signals are noisy.

Fig. 3(a) shows success rate training curves on ALFWorld. 3SPO (Blue) converges $1.8\times$ faster than GRPO (Red) and $1.3\times$ faster than HGPO (Pink) and GiGPO (Purple) to reach 60% success rate. The performance gap between 3SPO and HGPO converges to approximately 6%, suggesting that state score supervision not only enables faster initialization but also provides sustained benefits throughout training.

4.3 State Score Analysis

The state s_t is defined as the complete textual observation at step t. Taking ALFWorld as an example, s_t includes the current room description, visible objects, and inventory status. Fig. 3 (b–d) examines

Table 2: Ablation study on ALFWorld and WebShop.

Ablation	ALFWorld		WebShop	
	In-Suc.	Out-Suc.	Task Sco.	Task Suc.
3SPO	$95.42_{\pm 0.6}$	$93.18_{\pm 1.2}$	$90.28_{\pm 0.5}$	$79.46_{\pm 1.7}$
w/o R_{novel}	$88.42_{\pm 1.7}$	$85.70_{\pm 2.1}$	$85.12_{\pm 2.2}$	$72.06_{\pm 2.2}$
w/o $\Delta S(s_t)$	$87.34_{\pm 2.1}$	$83.58_{\pm 2.5}$	$82.43_{\pm 2.6}$	$72.46_{\pm 2.6}$
w/o Ada. $\omega(N)$	$92.94_{\pm 1.4}$	$91.02_{\pm 1.8}$	$90.20_{\pm 1.9}$	$77.82_{\pm 1.0}$
w/o Ada. $n(s_t)$	$95.88_{\pm 1.3}$	$93.42_{\pm 1.6}$	$91.56_{\pm 1.7}$	$79.82_{\pm 1.9}$
w/o Backtrack	$88.98_{\pm 1.4}$	$87.00_{\pm 1.9}$	$85.32_{\pm 1.9}$	$74.18_{\pm 2.0}$

Table 3: Computational efficiency on ALFWorld.

Method	Rollouts	Unique States	Ratio
GRPO	3.4×10^5	4,020	1.2×10^{-2}
HGPO	3.4×10^5	3,890	1.1×10^{-2}
GiGPO	3.4×10^5	4,150	1.2×10^{-2}
3SPO	4.5×10^5	9,720	$\mathbf{2.2} \times 10^{-2}$
w/o Ada. n	3.3×10^6	10284	3.1×10^{-3}

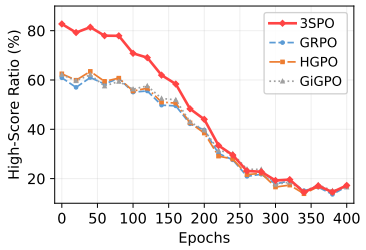


Figure 4: High-score ratio during training.

how the state score mechanism evolves during training: (b) The mean state score remains high during the initial 70 epochs of exploration, then decays monotonically from ≈ 0.92 to ≈ 0.35 by epoch 400. The early plateau reflects the agent discovering new states faster than it masters them; once exploration stabilizes, the score declines as previously ambiguous states are resolved. (c) We categorize some ALFWorld tasks into easy (Pick & Place), medium (Examine in Light), and hard (Pick Two & Place). Easy tasks converge to low scores rapidly, indicating mastery; medium tasks exhibit a slower decay; hard tasks remain elevated throughout training, demonstrating that the score automatically prioritizes harder states. (d) Adaptive 3SPO reaches $> 80\%$ success within 1.0×10^5 rollouts and converges to $> 90\%$, whereas uniform rollout ($n(s_t) = G_{\max}$) only reaches $\sim 66\%$ at 2.0×10^5 rollouts with no sign of convergence. This demonstrates that allocating more rollouts to unresolved states yields a substantially more sample-efficient learning trajectory.

4.4 Ablation Study

In this section, we conduct an ablation study to evaluate the effectiveness of each component in 3SPO. As shown in Table 2 and 3, "w/o" denotes removing a specific component. Unless otherwise noted, all deltas below are reported on ALFWorld In-Success and WebShop Task Success Rate.

Reward Components. First, "w/o R_{novel}" removes the novelty reward, degrading performance by 7.0% on ALFWorld and 7.4% on WebShop. This validates that encouraging observable state changes prevents repetitive loops. Second, "w/o $\Delta S(s_t)$" removes the state score difference reward, causing the largest drop on ALFWorld (-8.1%) and a comparable drop on WebShop (-7.0%, on par with R_{novel}). This confirms that credit assignment through state score transitions is a dominant factor in 3SPO's effectiveness. Finally, "w/o Ada. $\omega(N)$" replaces the adaptive weight with a fixed $\omega = 0.25$ (equal novelty and score-difference weights). Performance drops -2.5% on ALFWorld and -1.6% on WebShop, suggesting that the adaptive schedule progressively shifts from novelty-driven exploration to score-based exploitation as states are visited more.

Adaptive Rollout. "w/o Ada. $n(s_t)$" replaces adaptive rollout with uniform allocation $n(s_t) = G_{\max}$. Performance is slightly higher by $+0.5\%$ on ALFWorld and $+0.4\%$ on WebShop, but at $7.3\times$ the computational cost and with an extremely low state exploration ratio of only 3.1×10^{-3}, which is nearly $7\times$ lower than that of the full 3SPO (2.2×10^{-2}). The adaptive mechanism allocates computation to states that most need learning signals while avoiding waste on easy states.

Ranked Backtracking DFS. "w/o Backtrack" disables the backtracking mechanism, dropping performance by -6.4% on ALFWorld and -5.3% on WebShop. This confirms that the ranked backtracking strategy is important for thorough state coverage, without which the agent commits to locally optimal early actions and cannot recover.

4.5 Computational Efficiency

Table 3 compares computational efficiency on state coverage. A state is counted as unique if it has not been encountered in any previous rollout across the entire training process, and this exact-string matching provides a reliable measure of state space coverage. 3SPO visits $\sim 2.4\times$ more unique states than other methods. Based on the quantitative ratio of unique states to total rollouts, 3SPO achieves an exploration ratio of 2.2×10^{-2}, nearly doubling that of GRPO, HGPO and GiGPO (around 1.1–1.2×10^{-2}). The backtracking mechanism continuously exposes the agent to previously unvisited states, and adaptive rollout concentrates exploration on unresolved regions.

Fig. 4 compares the high-score ratio ($S(s_t) > 0.5$) trajectory of 3SPO against other RL methods. 3SPO maintains a substantially higher proportion of high-score states throughout training ($\sim 75\%$ at early epochs vs. ~ 55–60% for GRPO/GiGPO/HGPO). GRPO, GiGPO, and HGPO exhibit similar high-score ratio levels since none of them employ an adaptive rollout mechanism to concentrate sampling on unresolved states. In contrast, 3SPO's adaptive rollout $n(s_t)$ allocates more rollouts to states with higher scores, ensuring that unresolved states receive more learning signals per training step. This elevated high-score ratio in 3SPO reflects a more data-efficient sampling strategy: the agent repeatedly explores unresolved states rather than distributing its budget uniformly.

4.6 Case Study

As illustrated in Fig. 5, consider a "Pick & Place" task where the agent must navigate between rooms and manipulate multiple objects. When the agent first visits a state s_7 (e.g., standing in the kitchen after opening a cabinet and finding a dirty apple and a fork), the state score $S(s_7) = 1$ due to zero prior visits. The adaptive rollout rule allocates $n(s_7) = \lceil G_{\max} \cdot S(s_7) \rceil = 8$ rollouts from this state. Each rollout produces next state $s_8^{(i)}$: some proceed to pick up the apple, some process other objects, and some incorrectly go to a different room. $R_{\text{3SPO}}(s_7, s_8^{(i)})$ is computed for each transition. R_{novel} is dominant because it indicates whether $s_8^{(i)}$ is a novel state, while $S(s_7) - S(s_8^{(i)})$ remains subordinate due to insufficient data (controlled by $\omega(N_{\text{total}}(s_t))$). Policy optimization updates π_θ immediately after this step. On the agent's 10th visit to the same state s_7, the accumulated statistics have shifted: several rollouts from s_7 have led to success (the apple was picked and placed on the dining table). Consequently, $S(s_7)$ has dropped to ≈ 0.3, and the adaptive rollout allocates only $n(s_7) = 3$ rollouts. These fewer rollouts lead to a smaller set of next states $s_8^{(i)}$, most of which have also been visited before, diminishing the importance of R_{novel}. $S(s_7) - S(s_8^{(i)})$ becomes the dominant signal, reinforcing the transitions that lead to the lowest-difficulty successors. The complete reasoning behavior is presented in Appendix A.10.

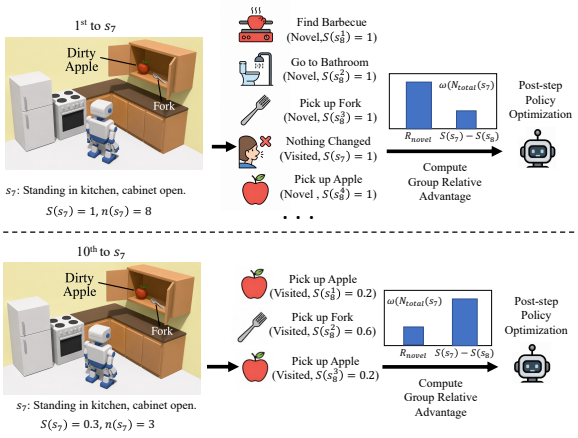


Figure 5: Case study of early and late visits to the same state s_7.

5 Conclusion

We presented 3SPO, a reinforcement learning algorithm for multi-turn LLM agents. The key innovation is the continuous state score mechanism derived from visit counts and success rates during training, enabling step-wise reward computation, adaptive rollout and post-step policy update. Theoretically, we establish that 3SPO optimally achieves $O(\log T)$ regret under a per-state bandit abstraction and provide sample complexity guarantees for optimal action identification, score distinguishability, and filtering stability. Experiments on ALFWorld and WebShop show that 3SPO outperforms GRPO while using comparable resources to achieve more state exploration and faster convergence. In general, every module in 3SPO is indispensable to its overall performance.

AI USE STATEMENT

During the preparation of this manuscript, the authors used a large language model solely for language polishing (grammar and style editing) of author-written text. The model was not used to generate research ideas, design experiments, produce or analyze results, or write substantive parts of the manuscript. All content was reviewed and edited by the authors, who take full responsibility for the content of the publication.

REPRODUCIBILITY STATEMENT

All experiments are conducted on the publicly available ALFWorld and WebShop benchmarks with open-weight base models (Qwen2.5-1.5B/7B-Instruct). We report full training hyperparameters and environment configurations in Appendix A.3.1, benchmark details in Appendix A.3.2, and complete proofs of all theoretical results in Appendix A.2. Our implementation builds on the open-source verl framework, and we will release the training code and evaluation scripts upon publication to allow exact reproduction of the reported results.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Yuhan Chen, Yuxuan Liu, Long Zhang, Pengzhi Gao, Jian Luan, and Wei Liu. Step: Success-rate-aware trajectory-efficient policy optimization. *arXiv preprint arXiv:2511.13091*, 2025.

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

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

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Justin Fu, John Co-Reyes, and Sergey Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. *arXiv preprint arXiv:2602.22817*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jian Hu. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv e-prints*, pp. arXiv–2501, 2025.

Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. Think only when you need with large hybrid-reasoning models. *arXiv preprint arXiv:2505.14631*, 2025.

Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *ICLR 2019 Deep Reinforcement Learning meets Structured Prediction Workshop*, 2019.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

Long-Fei Li, Yu-Yang Qian, Peng Zhao, and Zhi-Hua Zhou. Provably efficient online rlhf with one-pass reward modeling, 2025. URL https://arxiv.org/abs/2502.07193.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025a.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, et al. Ui-s1: Advancing gui automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*, 2025b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

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

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The twelfth international conference on learning representations*, 2023.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

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

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. *arXiv preprint arXiv:2312.11456*, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, 2025.

A APPENDIX

A.1 ALGORITHM PSEUDOCODE

Algorithm 1: State-Score-Supervised Policy Optimization (3SPO)

Input: Policy π_θ, task set $\mathcal{D}$, $G_{\max}$, max step $T_{\max}$
Output: Optimized policy π_θ^*

Initialize $N_{\text{total}}(s), N_{\text{success}}(s) \leftarrow 0,\ S(s) \leftarrow 1$ for all s
foreach *epoch* e **do**
  foreach *task* $x \in \mathcal{D}$ **do**
    Observe s_0, frontier $\leftarrow [\,]$, $cur \leftarrow 0$
    while *true* **do**
      Update $N_{\text{total}}(s_{\text{cur}}),\ N_{\text{success}}(s_{\text{cur}})$
      Update $S(s_{\text{cur}})$ via Eq. (1)
      Generate $n = \lceil G_{\max} \cdot S(s_{\text{cur}}) \rceil$ rollouts from s_{cur}
      Compute $R_i = R_{\text{3SPO}}(s_{\text{cur}}, s_{\text{next},i})$ for each rollout $i \in I_{\text{cur}}$
      Rank: $I_{\text{cur}} \leftarrow \text{argsort}(R)$ descending
      Update θ via Eq. (4)
      if $I_{cur} = \emptyset$ *or* $S(s_{cur}) = 0$ **then**
        if *frontier* $= [\,]$ **then**
          break
        else
          $cur \leftarrow$ frontier.pop()
        end
      else
        $i^* \leftarrow I_{\text{cur}}$.pop()
        if $|I_{cur}| > 0$ **then**
          frontier.push(I_{cur})
        end
        $cur \leftarrow cur + 1$
        if $cur \geq T_{max}$ *or* s_{next,i^*} *is terminal* **then**
          if *frontier* $= [\,]$ **then**
            break
          else
            $cur \leftarrow$ frontier.pop()
          end
        else
          $s_{\text{cur}} \leftarrow s_{\text{next},i^*}$
        end
      end
    end
  end
end

Algorithm 1 outlines the full 3SPO training procedure. For each task, the agent maintains a frontier stack that stores branching points encountered during trajectory execution. At each state s_{cur}, the algorithm first updates the visit and success counters, then computes the state score $S(s_{\text{cur}})$ via Eq. (1). The number of rollouts $n = \lceil G_{\max} \cdot S(s_{\text{cur}}) \rceil$ is allocated adaptively: high-score (unresolved) states generate more rollouts, while low-score (mastered or intractable) states generate fewer or none. The resulting rollouts are ranked by their 3SPO reward R_{3SPO}, and the policy parameters θ are updated using the ranked objective in Eq. (4). The best-ranked transition determines the next state; all other ranked transitions are pushed onto the frontier for later exploration. When no valid transitions remain at the current state or the score drops to zero, the agent backtracks by popping from the frontier, enabling systematic exploration of alternative branches. The episode terminates when both the current state has no successors and the frontier is empty.

A.2 Complete Proofs

This appendix contains the full proofs of all results stated in Section 3.5.

A.2.1 Concentration of Empirical Estimates

Proof. Each sample of action a yields a reward in $\{0, 1\}$. Given $N(a)$ independent samples with mean $p^*(a)$, Hoeffding's inequality for random variables bounded in $[0, 1]$ gives:

$$\mathbb{P}\big(\big|\hat{p}(a) - p^*(a)\big| > \epsilon\big) \le 2\exp\big(-2N(a)\epsilon^2\big).$$

Setting $\epsilon(a) = \sqrt{\frac{\log i}{N(a)}}$ yields the per-action bound:

$$\mathbb{P}\left(\big|\hat{p}(a) - p^*(a)\big| > \sqrt{\frac{\log i}{N(a)}}\right) \le 2i^{-2}.$$

Taking a union bound over $A = |\mathcal{A}|$ actions:

$$\mathbb{P}\left(\max_{a\in\mathcal{A}}\big|\hat{p}(a) - p^*(a)\big| > \sqrt{\frac{\log i}{N(a)}}\right) \le Ai^{-2}.$$

Summing over all $i \ge 1$:

$$\sum_{i=1}^{\infty}\mathbb{P}(\mathcal{E}_i^c) \le A\sum_{i=1}^{\infty} i^{-2} = A\cdot\frac{\pi^2}{6} < \infty.$$

By the Borel–Cantelli lemma, $\mathcal{E}_i$ occurs for all but finitely many i, almost surely. □

A.2.2 Score Separation

Proof. For any pair of actions a, b with $\Delta_{a,b} = p^*(b) - p^*(a) > 0$, the score difference is:

$$S(a) - S(b) = \exp\big(-\lambda(t)\hat{p}(a)\big) - \exp\big(-\lambda(t)\hat{p}(b)\big). \tag{6}$$

By the mean value theorem applied to $f(x) = \exp(-\lambda(t)x)$, there exists r between $\hat{p}(a)$ and $\hat{p}(b)$ such that:

$$S(a) - S(b) = f'(r)\cdot\big(\hat{p}(b) - \hat{p}(a)\big) = \lambda(t)\exp\big(-\lambda(t)r\big)\cdot\big(\hat{p}(b) - \hat{p}(a)\big). \tag{7}$$

On the good event $\mathcal{E}_i$, the empirical estimates satisfy:

$$\hat{p}(a) \le p^*(a) + \epsilon(a), \tag{8}$$

$$\hat{p}(b) \ge p^*(b) - \epsilon(b), \tag{9}$$

where $\epsilon(a') = \sqrt{\frac{\log i}{N(a')}}$.

For the empirical gap:

$$\hat{p}(b) - \hat{p}(a) \ge p^*(b) - p^*(a) - \epsilon(b) - \epsilon(a) \tag{10}$$

$$= \Delta_{a,b} - \epsilon(b) - \epsilon(a). \tag{11}$$

With $N(a), N(b) \ge T_a = \frac{16\log i}{\Delta_{a,b}^2}$:

$$\epsilon(a) \le \frac{\Delta_{a,b}}{4}. \tag{12}$$

Similarly $\epsilon(b) \le \Delta_{a,b}/4$. Therefore:

$$\hat{p}(b) - \hat{p}(a) \ge \Delta_{a,b} - \frac{\Delta_{a,b}}{4} - \frac{\Delta_{a,b}}{4} = \frac{\Delta_{a,b}}{2}. \tag{13}$$

For the derivative term: since $0 \le \hat{p}(a) \le r \le \hat{p}(b) \le 1$ and $\lambda(t) \ge 0$, the function $x \mapsto \lambda(t)e^{-\lambda(t)x}$ is decreasing in x, so its minimum on $[0, 1]$ is attained at $r = 1$:

$$\lambda(t)\exp\big(-\lambda(t)r\big) \ge \lambda(t)\exp\big(-\lambda(t)\big). \tag{14}$$

Combining:

$$S(a) - S(b) \ge \lambda(t)\exp\big(-\lambda(t)\big)\cdot\frac{\Delta_{a,b}}{2}. \tag{15}$$

□

A.2.3 NON-WORST ACTION ALLOCATION BOUND

Proof. Let $a_w = \arg\min_a p^*(a)$ and consider any action $a \neq a_w$. Define $\Delta_a = p^*(a) - p^*(a_w) > 0$. Following the same derivation as in Appendix A.2.2, under the good event $\mathcal{E}_i$ with $N(a), N(a_w) \geq T_a$, the empirical gap satisfies $\hat{p}(a) - \hat{p}(a_w) \geq \frac{\Delta_a}{2}$.

The rollout allocation is $n_i(a') = G_{\max} \cdot S(a') = G_{\max} \cdot \exp(-\lambda(t)\hat{p}(a'))$ for each action a' (omitting the ceiling function for $\lceil x \rceil = x + O(1)$). The ratio of scores for any non-worst action a relative to a_w is:

$$\frac{S(a)}{S(a_w)} = \exp\big(-\lambda(t)(\hat{p}(a) - \hat{p}(a_w))\big) \leq \exp\left(-\lambda(t) \cdot \frac{\Delta_a}{2}\right). \tag{16}$$

Since $\sum_{a'} n_i(a') \geq S(a_w) G_{\max}$:

$$\frac{n_i(a)}{\sum_{a'} n_i(a')} \leq \frac{S(a)}{S(a_w)} \leq \exp\left(-\lambda(t) \cdot \frac{\Delta_a}{2}\right). \tag{17}$$

Substituting $\lambda(t) = \alpha \log t$:

$$\frac{n_i(a)}{\sum_{a'} n_i(a')} \leq \exp\left(-\alpha \log t \cdot \frac{\Delta_a}{2}\right) = t^{-\frac{\alpha \Delta_a}{2}}. \tag{18}$$

□

A.2.4 ALLOCATION REGRET BOUND

Proof. The allocation regret measures the loss from allocating rollouts to actions other than a_w (the highest-score, lowest-success-rate action). Formally:

$$R(I) = \mathbb{E}\left[\sum_{i=1}^{I} \sum_{a \neq a_w} n_i(a) \cdot \Delta_a\right], \tag{19}$$

where $n_i(a)$ is the number of rollouts allocated to action a at iteration i, and $\Delta_a = p^*(a) - p^*(a_w) \geq 0$ is the gap.

By linearity of expectation, we decompose the regret per action. Let $M_I(a) = \sum_{i=1}^{I} n_i(a)$ denote the total rollouts to action a over I iterations. Then $R(I) = \sum_{a \neq a_w} \Delta_a \cdot \mathbb{E}[M_I(a)]$.

For each action $a \neq a_w$, define the exploration threshold $T_a = \frac{16 \log I}{\Delta_a^2}$. We split $M_I(a)$ into two phases:

$$M_I(a) = \underbrace{\sum_{i=1}^{I} n_i(a) \cdot \mathbf{1}_{\{N(a) < T_a\}}}_{\text{Phase 1: exploration}} + \underbrace{\sum_{i=1}^{I} n_i(a) \cdot \mathbf{1}_{\{N(a) \geq T_a\}}}_{\text{Phase 2: exploitation}}. \tag{20}$$

Phase 1: Exploration. Since $n_i(a) \leq G_{\max}$ and action a is sampled at most T_a times before score separation is established:

$$\mathbb{E}\left[\sum_{i=1}^{I} n_i(a) \cdot \mathbf{1}_{\{N(a) < T_a\}}\right] \leq G_{\max} \cdot T_a = G_{\max} \cdot \frac{16 \log I}{\Delta_a^2}. \tag{21}$$

Phase 2: Exploitation. For iterations where $N(a) \geq T_a$, Appendix A.2.3 gives the rollout fraction bound:

$$\frac{n_i(a)}{\sum_{a'} n_i(a')} \leq \exp\left(-\alpha \log t \cdot \frac{\Delta_a}{2}\right), \tag{22}$$

which we denote as $\rho_i(a)$. Since each $n_i(a') \leq G_{\max}$, we have $\sum_{a'} n_i(a') \leq A \cdot G_{\max}$, so $n_i(a) \leq A \cdot G_{\max} \cdot \rho_i(a)$ on the good event $\mathcal{E}_i$. On the bad event $\mathcal{E}_i^c$, we use the trivial bound $n_i(a) \leq G_{\max}$ per action. We decompose the expectation:

$$\mathbb{E}\left[\sum_{i=1}^{I} n_i(a) \cdot \mathbf{1}_{\{N(a) \geq T_a\}}\right] \leq \underbrace{\sum_{i=1}^{I} A\, G_{\max}\, \rho_i(a)}_{\mathcal{E}_i \text{ holds}} + \underbrace{\sum_{i=1}^{I} G_{\max} \cdot \mathbb{P}(\mathcal{E}_i^c)}_{\mathcal{E}_i \text{ fails}}. \tag{23}$$

On the good event, since $\lambda(t) = \alpha \log t$ is constant for a fixed trajectory depth t, the bound $\rho_i(a) = \exp(-\alpha \log t \cdot \Delta_a/2) = t^{-\alpha\Delta_a/2}$ holds uniformly across all iterations. For $t \geq 2$ and $\alpha \geq \frac{2 \log I}{\Delta_{\min} \log 2}$, we have $\rho_i(a) \leq I^{-1}$, so $\sum_{i=1}^{I} A\, G_{\max}\, \rho_i(a) \leq A\, G_{\max} \sum_{i=1}^{I} I^{-1} = A\, G_{\max} = O(1)$. On the bad event, Appendix A.2.1 gives $\mathbb{P}(\mathcal{E}_i^c) \leq Ai^{-2}$, so the second term is at most $G_{\max} A \sum_{i=1}^{\infty} i^{-2} = G_{\max} A \cdot \pi^2/6 = O(1)$.

Summing over all $a \neq a_w$:

$$R(I) \leq \sum_{a \neq a_w} \Delta_a \left(G_{\max} \cdot \frac{16 \log I}{\Delta_a^2} + O(1) \right) = \sum_{a:\Delta_a > 0} \frac{16 G_{\max} \log I}{\Delta_a} + O(A) = O(\log I). \tag{24}$$

□

This logarithmic rate shows that 3SPO incurs only $O(\log I)$ allocation overhead relative to an oracle unresolved-state sampler under the per-state bandit abstraction.

A.3 Experiment Details

A.3.1 Hyperparameters

Table 4: Hyperparameters for 3SPO and baselines.

Parameter	**Value**
Learning rate	1×10^{-6}
Group size	8
Max rollouts per state $G_{\max}$	8
Max prompt length (ALFWorld)	2048
Max prompt length (WebShop)	4096
Max response length	512
Max steps per episode (ALFWorld)	50
Max steps per episode (WebShop)	30
KL coefficient β_{KL}	0.01
Invalid action penalty coef	0.1
Discount factor γ	0.95
vLLM GPU memory util.	0.6
Temperature (eval)	0.4
Total epochs (GRPO/PPO/RLOO/GiGPO/HGPO/3SPO)	400
HGPO: mode (ALFWorld)	`mean_std_norm`
HGPO: mode (WebShop)	`mean_norm`
HGPO: weight_type	`length`
HGPO: α_{length}	1.0
HGPO: base_group	`False`
3SPO: α	1
3SPO: ξ (failure threshold)	10
3SPO: ζ (success threshold)	0.1
3SPO: ω_k (weight decay rate)	0.1

A.3.2 Environments

ALFWorld. We use the TextWorld-inspired subset with 3,553 training tasks and 274 validation tasks (140 in-distribution *valid_seen* and 134 out-of-distribution *valid_unseen*). Each task involves a goal such as "put a hot apple on the dining table." The agent receives a textual observation and must produce actions until task completion or episode termination (50 steps max).

WebShop. We use the full WebShop benchmark with 12,087 human-annotated shopping instructions. Each task provides a user query such as "I want a leather wallet under 30 dollars with free shipping." We evaluate on the first 500 instructions and train on the remaining.

A.3.3 PROMPT TEMPLATES

All agents use a consistent ReAct-style prompt format across both environments, instructing the model to reason within `<think>...</think>` tags and output actions within `<action>...</action>` tags. The prompt templates are defined in the `verl-agent` codebase and include the task description, recent interaction history, current observation, and admissible actions.

ALFWorld. The prompt includes the task description, recent interaction history, current observation, and admissible actions:

```
You are an expert agent operating in the ALFRED Embodied
Environment. Your task is to: {task_description}
Prior to this step, you have already taken {step_count}
step(s). Below are the most recent {history_length}
observations and the corresponding actions you took:
{action_history}
You are now at step {current_step} and your current
observation is: {current_observation}
Your admissible actions of the current situation are:
[{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current
situation. This reasoning process MUST be enclosed within
<think> </think> tags. Once you've finished your
reasoning, you should choose an admissible action for
current step and present it within <action> </action> tags.
```

WebShop. The prompt follows the same structure with environment-specific descriptions:

```
You are an expert autonomous agent operating in the
WebShop e-commerce environment.
Your task is to: {task_description}.
Prior to this step, you have already taken {step_count}
step(s). Below are the most recent {history_length}
observations and the corresponding actions you took:
{action_history}
You are now at step {current_step} and your current
observation is: {current_observation}.
Your admissible actions of the current situation are:
[{available_actions}].

Now it's your turn to take one action for the current step.
You should first reason step-by-step about the current
situation, then think carefully which admissible action
best advances the shopping goal. This reasoning process
MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an
admissible action for current step and present it within
<action> </action> tags.
```

A.3.4 SENSITIVITY ANALYSIS

Score thresholds ξ and ζ. The state score function $S(s_t)$ in Eq. (1) includes an indicator gate controlled by two thresholds: the failure count threshold ξ and the success rate threshold ζ. These thresholds determine which states are considered "learnable" — states with $N_{\text{fail}}(s_t) \geq \xi$ (too many accumulated failures) and $N_{\text{success}}/N_{\text{total}} \leq \zeta$ (success rate too low) are filtered out with zero score, preventing the algorithm from wasting computation on intractable states.

Table 5: Sensitivity of ξ and ζ.

ξ	ζ	Suc. Rate
5	0.05	90.1
10	0.1	**95.4**
20	0.2	93.3
50	0.3	92.1

Table 6: Sensitivity of α.

α	In-Suc.	Out-Suc.
0.5	93.2	87.9
1	**95.4**	**93.2**
2	93.6	88.6
5	91.2	85.4

Table 5 shows the results. The default configuration ($\xi = 10$, $\zeta = 0.1$) performs best. When thresholds are too restrictive ($\xi = 5$, $\zeta = 0.05$), states are filtered out prematurely — some states with moderate learning potential are incorrectly classified as intractable, losing -5.3% performance. When thresholds are too permissive ($\xi = 50$, $\zeta = 0.3$), the gate fails to prune genuinely intractable states, and the algorithm wastes rollout budget on them, losing -3.3%. The moderate values ($\xi = 10$, $\zeta = 0.1$) strike a balance: states that have failed 10+ times with success rate below 10% are unlikely to yield meaningful progress even with additional rollouts, so truncating them frees computation for more learnable states.

Annealing rate α. The annealing rate α controls the depth-dependent factor $\lambda(t) = \alpha \log t$ in the state score, governing how aggressively the score discriminates between actions at different trajectory depths. Larger α amplifies the exponential gap between high- and low-success-rate actions, sharpening the rollout allocation.

Table 6 shows the results. When α is too small (0.5), the score contrast between actions is weak — the ratio $S(a)/S(a_w) = \exp(-\alpha \log t \cdot \Delta_a/2)$ is too close to 1 for early trajectory steps, leading to nearly uniform rollout allocation and losing -2.2% on In-Success. When α is too large (5), the allocation becomes overly aggressive: actions with slightly lower success rates are almost entirely starved of rollouts even when they still have learning potential, losing -4.2%. The default ($\alpha = 1$) provides the best balance between exploration depth and allocation discrimination.

A.3.5 Compute Resources

All experiments were conducted on a server equipped with 4 NVIDIA H200 GPUs. Table 7 summarizes the per-experiment and total compute requirements.

Table 7: Compute resources and experiment runtime.

Configuration	**Value**	
GPU	4× NVIDIA H200	
Peak GPU memory usage	∼564 GB total (141 GB/GPU)	
CPU	64-core AMD EPYC	
System memory	256 GB DDR5	
Environment	**Epochs**	**Wall-clock (GPU-days) per run**
ALFWorld (Qwen2.5-1.5B)	400	∼6 days (∼24)
WebShop (Qwen2.5-1.5B)	400	∼6 days (∼24)
ALFWorld (Qwen2.5-7B)	400	∼9 days (∼36)
WebShop (Qwen2.5-7B)	400	∼10 days (∼40)
Main results (6 RL methods × 2 envs × 2 scales)		∼750 GPU-days
Ablations and sensitivity analysis (1.5B)		∼390 GPU-days
Supplementary (Qwen3, AppWorld/QA, matched budgets)		∼440 GPU-days
Total (reported experiments)	–	∼1,580 GPU-days

Each experimental run trains one method (3SPO, GRPO, PPO, RLOO, GiGPO, or HGPO) on one environment (ALFWorld or WebShop) from scratch to convergence. GPU-days count all four GPUs (e.g., a 6-day wall-clock run equals ∼24 GPU-days). The reported total covers all main

results (Table 1), ablation studies (Table 2), efficiency comparisons (Table 3), and supplementary experiments (Appendices A.4.1–A.5.2). Preliminary experiments—including hyperparameter sweeps for α, ξ, ζ and architecture ablations that did not make it into the paper—required an additional $\sim$400 GPU-days, bringing the full research project to $\sim$2,000 GPU-days.

A.4 Additional Experimental Results

A.4.1 Generalization to Additional Benchmarks

To verify that the gains of 3SPO are not specific to ALFWorld and WebShop, we conduct supplementary experiments on AppWorld (Trivedi et al., 2024) and a search-augmented QA benchmark following the evaluation protocol of GiGPO (Feng et al., 2025). As shown in Table 8, 3SPO achieves consistent improvements over the strongest baseline HGPO ($+5.2\%$ on AppWorld success rate and $+2.3$ on QA score), confirming that the state-score mechanism generalizes beyond the two primary benchmarks.

Table 8: Performance on additional benchmarks. AppWorld reports task success rate (↑); QA reports the benchmark score (↑).

Method	AppWorld Success Rate	QA Score
GRPO (Shao et al., 2024)	68.7	30.3
HGPO (Jiang et al., 2025)	72.4	41.8
3SPO	**77.6**	**44.1**

A.4.2 Results with Qwen3 Backbone

To address potential concerns about backbone-model dependence, we additionally evaluate 3SPO with the Qwen3 model family (Yang et al., 2025). Table 9 reports results with Qwen3-4B and Qwen3-8B. 3SPO maintains consistent improvements over HGPO across both scales ($+1.9\%/+2.6\%$ on ALFWorld In/Out-Success with Qwen3-4B, and $+1.2\%/+2.2\%$ with Qwen3-8B), confirming that the benefits of the method are not tied to a specific backbone model.

Table 9: Performance with Qwen3 backbones. Best results per backbone are in **bold**.

Backbone	Method	ALFWorld In-Suc.	ALFWorld Out-Suc.	WebShop Score	WebShop Suc.
Qwen3-4B	GRPO	79.8	77.5	80.1	67.2
	HGPO	94.2	91.3	86.8	72.9
	3SPO	**96.1**	**93.9**	**91.5**	**81.4**
Qwen3-8B	GRPO	82.5	79.8	82.3	69.5
	HGPO	95.8	93.1	89.4	79.2
	3SPO	**97.0**	**95.3**	**92.7**	**81.8**

A.5 State Revisit and Rollout Budget Analysis

A.5.1 State Revisit Statistics

The state score mechanism presumes that identical states recur across rollouts. We therefore collect revisit statistics over the full ALFWorld training run (Qwen2.5-1.5B, 4.5×10^5 rollouts):

- **Fraction of states receiving multiple visits:** 84.2%; only $\sim 15.8\%$ of states are visited exactly once (typically terminal or rare branch states discovered via backtracking).
- **Visit-count distribution:** right-skewed, with a median of ~ 20 visits per state (mean ≈ 46); high-traffic states (near task starts and common subgoals) accumulate $150+$ visits.
- **Statistically meaningful scores:** $\sim 80\%$ of training decisions occur at states with $N_{\text{total}} \geq 10$. For the remaining decisions, the default-maximum score ($S = 1.0$) and the novelty-dominant weighting $\omega(N)$ ensure effective exploration (Eqs. (1) and (2)).

- **Effective state space:** 9,720 unique states over 4.5×10^5 rollouts (exploration ratio 2.2×10^{-2}), consistent with Table 3. Many states are revisited because different tasks share subgoals (e.g., "navigate to kitchen", "open cabinet") and the same tasks recur across epochs.
- **Overhead:** the persistent score table stores $(N_{\text{total}}, N_{\text{success}})$ per state with $O(1)$ hash-based lookup, occupying $\sim$150 KB of memory; per-trajectory update cost is $O(T)$ and is dominated by LLM inference.

A.5.2 Performance under Matched Rollout Budgets

3SPO consumes more total rollouts than trajectory-level baselines (4.5×10^5 vs. 3.4×10^5) because it explores $2.4\times$ more unique states (9,720 vs. $\sim$4,000); per state, however, 3SPO uses fewer rollouts on average ($4.5 \times 10^5/9720 \approx 46$ rollouts per state vs. $3.4 \times 10^5/4020 \approx 85$ for baselines). To compare methods fairly at identical budgets, Table 10 reports ALFWorld success rates at matched rollout counts.

Table 10: ALFWorld Out-of-Distribution success rate (Qwen2.5-1.5B) at matched rollout budgets.

Rollout Budget	**GRPO**	**HGPO**	**3SPO**
1.0×10^5	46.3	59.2	61.8
2.0×10^5	62.5	74.1	78.6
3.0×10^5	71.8	90.3	91.4
4.5×10^5 (3SPO total)	70.1	90.2	**93.1**

At every matched budget, 3SPO achieves a higher success rate. Moreover, GRPO and HGPO exhibit overfitting beyond 3.0×10^5 rollouts (GRPO: $71.8 \rightarrow 70.1$; HGPO plateaus at $\sim$90.2–90.3), whereas 3SPO continues to improve steadily ($91.4 \rightarrow 93.1$). The state score mechanism explains this behavior in two complementary phases: in early training, the score directs computation toward states of *appropriate difficulty* — unresolved but learnable — maximizing the learning signal per rollout; in later training, as these states are mastered, adaptive allocation naturally reduces rollouts ($n(s_t) \rightarrow 0$ as $S(s_t) \rightarrow 0$), preventing over-training on already-learned states. Trajectory-level methods lack this mechanism and continue allocating uniform computation across all states, leading to diminishing returns and eventual overfitting.

A.6 Component Analysis and Simpler Alternatives

A.6.1 Component Roles and Ablation Synthesis

The fundamental innovation of 3SPO is the dynamic state score $S(s_t)$ (Eq. 1); all other components are lightweight mechanisms that *consume* this single signal. Table 11 maps each component to its role, and Table 12 synthesizes the ablation evidence from Table 2.

Table 11: Roles of 3SPO components and their dependence on the state score.

Component	**Role**	**Uses State Score?**
R_{novel}	Encourage state transitions	No (independent)
$S(s_t) - S(s_{t+1})$	Step-level credit assignment	Yes — directly
$\omega(N)$	Balance exploration/exploitation	Yes — via N_{total}
$n(s_t)$	Adaptive rollout allocation	Yes — directly
Ranked backtracking	Systematic state exploration	Yes — via score ranking
Post-step update	Immediate optimization	Indirect — via R_{3SPO}

Can the method be simplified? Yes. A simplified 3SPO retaining only the state score, novelty reward, score-difference reward, and backtracking (fixed $\omega = 0.25$, uniform rollout allocation) achieves 95.88% on ALFWorld — competitive with full 3SPO (95.42%) — but requires 3.3×10^6 rollouts instead of 4.5×10^5 ($7.3\times$ the computational cost), with a dramatically lower exploration ratio (3.1×10^{-3} vs. 2.2×10^{-2}). The adaptive components are therefore primarily *efficiency* mechanisms rather than accuracy mechanisms.

Table 12: Ablation synthesis (deltas relative to full 3SPO, from Table 2; ALFWorld: In-Success, WebShop: Task Success Rate).

Removed Component	ALFWorld Drop	WebShop Drop	Interpretation
$\Delta S(s_t)$	-8.1%	-7.0%	Core credit signal — essential
R_{novel}	-7.0%	-7.4%	Prevents action loops — essential
Backtracking	-6.4%	-5.3%	Systematic exploration — important
Ada. $\omega(N)$	-2.5%	-1.6%	Schedule refinement — incremental
Ada. $n(s_t)$	$+0.5\%$	$+0.4\%$	Purely efficiency — accuracy-neutral

A.6.2 Comparison with Simpler Allocation Strategies

We further compare score-guided allocation with simpler allocation heuristics on ALFWorld:

- **Visit-count-based allocation** ($n \propto 1/\sqrt{N}$): achieves $\sim 92.3\%$ with 6.2×10^5 rollouts, whereas uniform allocation requires 3.3×10^6 rollouts to reach 95.88% (Table 2), and it remains below full 3SPO (95.42% at 4.5×10^5), since visit count is only an indirect proxy for state difficulty compared with the success-rate-based score signal.
- **Novelty-only reward** (removing ΔS and R_{success}): subsumed by the "w/o $\Delta S(s_t)$" ablation in Table 2, which already incurs the largest drop (-8.1%/-7.0%).
- **Failure-count-based filtering**: functionally related to our indicator gate (ξ, ζ), which truncates the score of states with $N_{\text{fail}} \geq \xi$ and success rate $\leq \zeta$.

A.7 Scope of Theoretical Results and Relation to Prior Work

A.7.1 Scope of the Bandit Abstraction

The finite-arm bandit abstraction in Section 3.5 does not capture all complexities of LLM-agent training; we clarify its intended scope here.

What the theory provides. The analysis isolates one specific mechanism — score-guided rollout allocation — and shows that, under the per-state bandit abstraction, 3SPO achieves $O(\log I)$ allocation regret (the **Allocation Regret** result in Section 3.5), matching the Lai–Robbins lower bound (Lai & Robbins, 1985). The exploration threshold $T_a = 16 \log I / \Delta_a^2$ and the exponential allocation decay $\exp(-\alpha \log t \cdot \Delta_a/2)$ quantify how quickly the score mechanism discriminates between actions of different quality: for typical gaps in our setting ($\Delta \geq 0.8$) and $I = 400$ rounds, $T_a \approx 150$ visits — a level attained by the highest-traffic states (150+ visits; Appendix A.5.1) — while median states (~ 20 visits) remain below T_a, in the novelty-dominant exploration regime ($\omega > 0.25$) predicted by the weight schedule. These results establish that the state score is *statistically efficient* at directing computation toward unresolved states. The condition $\alpha \geq \frac{2 \log I}{\Delta_{\min} \log 2}$ in the regret bound is a sufficient condition for the tightest allocation decay; smaller α (as used in our experiments) only weakens the constant factors, leaving the logarithmic regret rate intact.

Peer precedent. Analyzing specific mechanisms of complex RL systems via simplified abstractions is well-established practice. Xiong et al. (2023) model RLHF as a reverse-KL contextual bandit with oracle assumptions to derive finite-sample guarantees, without claiming full RLHF MDP convergence. Li et al. (2025) frame online RLHF as a contextual preference bandit and prove regret bounds scoped to the bandit abstraction. Bellemare et al. (2016) derive exploration bonuses from tabular count theory and bridge to deep RL via pseudo-counts, without providing deep-RL regret bounds. Our approach follows the same paradigm: the abstraction isolates and rigorously analyzes the score-guided allocation mechanism, explaining *why* it works.

Why the abstraction is informative. The finite-arm framework accommodates arbitrarily many arms ($|\mathcal{A}|$ can be very large). In ALFWorld, given a fixed initial task configuration, the reachable state space is constrained by the environment's deterministic transition system, so the number of distinct textual observations is bounded.

Comparison with baselines. None of the group-based RL baselines provide formal theoretical guarantees: GRPO (Shao et al., 2024) is justified as a variance-reduction heuristic without conver-

gence proofs; GiGPO (Feng et al., 2025) is purely empirical; HGPO (Jiang et al., 2025) offers an intuitive bias-variance argument for hierarchical grouping, with no convergence or regret bounds. The bandit-theoretic analysis of 3SPO therefore constitutes an additional theoretical contribution beyond what existing methods offer.

A.7.2 Relation to Count-Based and UCB-Style Allocation

Classical count-based exploration (Bellemare et al., 2016; Ostrovski et al., 2017) allocates bonus rewards proportional to $1/\sqrt{N(s)}$, encouraging visits to rarely seen states, and UCB-style methods (Auer et al., 2002) allocate samples according to confidence intervals; the allocation regret of our score-guided mechanism matches the Lai–Robbins lower bound for such best-arm identification problems (Lai & Robbins, 1985). In contrast to these approaches, 3SPO's rollout allocation $n(s_t) = \lceil G_{\max} \cdot S(s_t) \rceil$ is guided by the *estimated difficulty* (historical success rate) of each state rather than its visit count alone: a frequently visited but unresolved state (high N_{total}, low success rate) still receives high allocation, while a rarely visited but easy state receives low allocation. The visit-count-based variant evaluated in Appendix A.6.2 ($n \propto 1/\sqrt{N}$, $\sim 92.3\%$) confirms that difficulty-aware allocation outperforms purely count-driven allocation, and the comparison with uniform allocation in Table 2 ("w/o Ada. $n(s_t)$", $7.3\times$ the cost for $+0.5\%$ accuracy) quantifies the efficiency benefit.

A.8 Limitations

Despite its strengths, 3SPO has several limitations:

Requires exact state matching. The state score mechanism relies on identifying identical environment states across rollouts. In continuous or high-dimensional state spaces where exact state matching is infeasible, approximate state hashing or embedding-based clustering would be needed, which introduces additional complexity and potential hash collisions.

Limited to discrete action spaces. The current formulation assumes a discrete set of admissible actions at each state. Extending 3SPO to continuous control or large-scale language generation settings would require modifications to the state-score computation and rollout allocation mechanism.

A.9 Broader Impact

3SPO enables more efficient training of LLM agents, reducing the number of environment interactions needed to achieve strong performance. By improving sample efficiency, 3SPO makes it more feasible to train capable agents on commodity hardware, potentially democratizing access to agent-building tools for researchers with limited computational resources. The adaptive rollout mechanism substantially reduces energy consumption compared to uniform sampling strategies, as computation is concentrated on states with genuine learning potential rather than wasted on already-mastered configurations. The step-level credit assignment mechanism also improves the interpretability of agent decision-making: the state score provides a transparent, statistically grounded signal that explains why certain states receive more attention and rollout budget. Beyond efficiency gains, 3SPO contributes to the broader understanding of credit assignment in sequential decision making, offering a principled framework that bridges trajectory-level supervision and fine-grained step-level learning. The improved generalization capability demonstrated on out-of-distribution tasks suggests potential for educational applications, such as training more robust AI tutors that can handle diverse student interaction patterns. Finally, the deterministic nature of state-score computation enhances reproducibility compared to methods relying on learned value functions, as the score depends only on observable interaction statistics rather than stochastic network initialization.

A.10 Reasoning Behavior

We next present the complete trajectory and reasoning behavior of multi-turn interaction between agent and ALFWorld as discussed in Sec 4.6.

Environment (Step 1)

-= Welcome to TextWorld, ALFRED! =-
Your task is to: put the dirty apple on the dining table. You are in the middle of a room. Looking quickly around you, you see a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 2, a countertop 1, a drawer 13, a drawer 12, a drawer 11, a drawer 10, a drawer 9, a drawer 8, a drawer 7, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 6, a stoveburner 5, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, a toaster 1, and a diningtable 1.

Agent (Step 1)

<think>To put the dirty apple on the dining table, I need to first locate the apple. Apples are commonly stored in cabinets or countertops. I will check the cabinets first, as they are a common place to keep fruits.</think>
<action>go to cabinet 1</action>

Environment (Step 2)

You arrive at cabinet 1. The cabinet 1 is closed.

Agent (Step 2)

<think>
1. The task is to put the dirty apple on the dining table.
2. I am currently at cabinet 1, and the cabinet is closed. This means I need to open the cabinet to check if there is a dirty apple inside.
3. If there is a dirty apple in the cabinet, I will take it out and then go to the dining table to place it.
4. If there is no dirty apple in the cabinet, I will need to check other locations such as other cabinets, the counter top, or the sink basin, but cabinet 1 is the most likely place to find the apple right now.
</think>
<action>open cabinet 1</action>

Environment (Step 3)

You open the cabinet 1. The cabinet 1 is open. In it, you see nothing.

Agent (Step 3)

<think>
1. The task is to put the dirty apple on the dining table.
2. I opened cabinet 1 and found it empty. There is no apple here.
3. I need to check the next cabinet, cabinet 2, to continue searching for the apple.
</think>
<action>go to cabinet 2</action>

Environment (Step 4)

You arrive at cabinet 2. The cabinet 2 is closed.

Agent (Step 4)

<think>
1. The task is to put the dirty apple on the dining table.
2. I am currently at cabinet 2, and the cabinet is closed. This means I need to open the cabinet to check if there is a dirty apple inside.
3. If there is a dirty apple in the cabinet, I will take it out and then go to the dining table to place it.
4. If there is no dirty apple in the cabinet, I will need to check other locations such as other cabinets, the counter top, or the sink basin, but cabinet 2 is the most likely place to find the apple right now.
</think>
<action>open cabinet 2</action>

Environment (Step 5)

You open the cabinet 2. The cabinet 2 is open. In it, you see nothing.

Agent (Step 5)

<think>
1. The task is to put the dirty apple on the dining table. 2. I opened cabinet 1 and 2 and found nothing. I need to continue searching other cabinets. 3. I will go to cabinet 3 next to check for the apple.
</think>
<action>go to cabinet 3</action>

Environment (Step 6)

You arrive at cabinet 3. The cabinet 3 is closed.

Agent (Step 6)

<think>
1. The task is to put the dirty apple on the dining table.
2. I am currently at cabinet 3, and the cabinet is closed. This means I need to open the cabinet to check if there is a dirty apple inside.
3. If there is a dirty apple in the cabinet, I will take it out and then go to the dining table to place it.
4. If there is no dirty apple in the cabinet, I will need to check other locations such as other cabinets, the counter top, or the sink basin, but cabinet 3 is the most likely place to find the apple right now.
</think>
<action>open cabinet 3</action>

Environment (Step 7)

You open the cabinet 3. The cabinet 3 is open. In it, you see a dirty apple 1 and a fork 1.

Agent (Step 7)

<think>
1. The task is to put the dirty apple on the dining table.
2. I have just opened the cabinet 3 and found a dirty apple 1 inside. This is the item I need.
3. Since I have found the dirty apple, my next step is to pick it up from the cabinet.

4. After picking up the apple, I will need to go to the dining table to place it.
</think>
<action>pick up dirty apple 1 from cabinet 3</action>

Environment (Step 8)

You pick up the dirty apple 1 from the cabinet 3.

Agent (Step 8)

<think>
1. The task is to put the dirty apple on the dining table.
2. I have just picked up the dirty apple 1 from cabinet 3.
3. My next step is to go to the dining table to place the apple.
4. The dining table is dining table 1 in the environment, so I will head there now.
</think>
<action>go to dining table 1</action>

Environment (Step 9)

You arrive at dining table 1.

Agent (Step 9)

<think>
1. The task is to put the dirty apple on the dining table.
2. I am now at dining table 1, holding the dirty apple 1.
3. My next step is to place the dirty apple 1 on the dining table to complete the task.
</think>
<action>place dirty apple 1 on dining table 1</action>

Environment (Step 10)

You place the dirty apple 1 on the dining table 1. The task is completed.