Verified Critical Step Optimization for LLM Agents

Anonymous ACL submission

Abstract

As large language model agents tackle increasingly complex long-horizon tasks, effective post-training becomes critical. Prior work faces fundamental challenges: outcome-only rewards fail to precisely attribute credit to intermediate steps, estimated step-level rewards introduce systematic noise, and Monte Carlo sampling approaches for step reward estimation incur prohibitive computational cost. Inspired by findings that only a small fraction of high-entropy tokens drive effective RL for reasoning, we propose **Critical Step Optimization (CSO)**, which focuses preference learning on *verified critical steps*, decision points where alternate actions demonstrably flip task outcomes from failure to success. Crucially, our method starts from failed policy trajectories rather than expert demonstrations, directly targeting the policy model's weaknesses. We use a process reward model (PRM) to identify candidate critical steps, leverage expert models to propose high-quality alternatives, then continue execution from these alternatives using the policy model itself until task completion. Only alternatives that the policy successfully executes to correct outcomes are verified and used as DPO training data, ensuring both quality and policy reachability. This yields fine-grained, verifiable supervision at critical decisions while avoiding trajectory-level coarseness and step-level noise. Experiments on GAIA-Text-103 and XBench-DeepSearch show that CSO achieves 37% and 26% relative improvement over the SFT baseline and substantially outperforms other post-training methods, while requiring supervision at only 16% of trajectory steps. This demonstrates the effectiveness of selective verification-based learning for agent post-training.

1 Introduction

The rapid advancement of large language models (LLMs), especially those tailored for multi-step reasoning and decision-making, such as GPT-5 (OpenAI, 2024), Claude 4.5 (Anthropic, 2025), Kimi-K2 (Kimi-Team, 2025), DeepSeek-R1 (DeepSeek-AI, 2025) and DeepSeek-V3.2 (dee, 2025), has laid the foundation for increasingly capable agentic systems that solve complex tasks through iterative tool use and environment interaction. As these LLM-based agents tackle more long-horizon, multi-stage problems, effective post-training techniques become critical for translating general model capabilities into robust and aligned behavior in deployment scenarios. Recent pipelines of agent post-training (Li et al., 2025a; Wu et al., 2025a) adopt a two-stage paradigm: supervised fine-tuning (SFT)

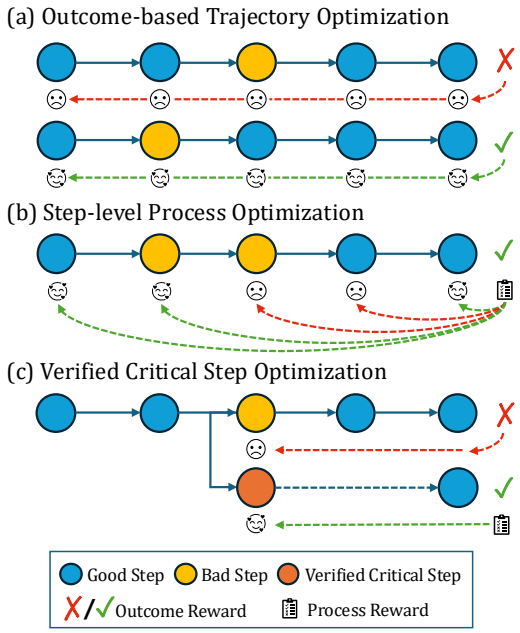

Figure 1: **Comparing post-training paradigms for LLM agents.** (a) Trajectory-level optimization assigns coarse outcome rewards to entire trajectories, while (b) step-level process optimization relies on intermediate rewards which could be inaccurate. (c) Verified Critical Step Optimization (CSO) focuses learning on verified critical steps where alternative actions demonstrably flip task outcomes, providing precise credit assignment with minimal supervision.

followed by reinforcement learning (RL). However, SFT often suffers from off-policy distribution shift (Tao et al., 2025; Wang et al., 2025b). While on-policy RL mitigates this, it requires high computational cost from full rollouts and suffers from sparse and delayed reward signals that hinder credit assignment (Wu et al., 2025a; Wei et al., 2025).

Recent work jumping out of the scope of SFT and online RL has explored *offline or semi-online post-training* approaches that balance efficiency and performance. As shown in Figure 1, one line focuses on *outcome-based trajectory optimization*. Methods like ETO (Song et al., 2024) and MiroThinker (MiroMind, 2025) build preference pairs from successful and failed trajectories, offering scalability but suffering from imprecise credit assignment in which reasonable actions in failed trajectories are penalized uniformly, while suboptimal decisions in successful ones are reinforced. Another line pursues *step-level process optimization*. AgentRPM (Choudhury, 2025) trains process reward models for intermediate assessments, while Co-Evolving Agents (Jung et al., 2025) uses failure agents to generate hard negatives. Though finer-grained, these methods rely on estimated process rewards that introduce noise and systematic biases. Hybrid approaches like IPR (Xiong et al., 2024) attempt to combine both paradigms but face significant practical challenges. IPR derives step-level rewards through Monte Carlo sampling from each step forward, which becomes prohibitively expensive on complex tasks. From another perspective, existing methods overlook that *not all steps are equally important*, as many involve trivial actions or few alternatives. Recent work on RLVR, particularly entropy-oriented algorithms (Wang et al., 2025c), shows that only a small fraction of high-entropy tokens drive effective RL for reasoning. This suggests focusing on *critical branching points* where alternate actions drastically change outcomes, rather than uniform or dense supervision.

We propose **Critical Step Optimization (CSO)**, which focuses preference learning on *verified critical steps* where alternate actions change the task outcome from failure to success. Unlike trajectory-level methods that apply coarse rewards uniformly, CSO provides fine-grained, step-specific supervision. Unlike step-level methods relying on estimated rewards, CSO grounds supervision in verified outcomes, eliminating process reward noise. Inspired by the high-entropy token principle in Wang et al. (2025c), we identify critical steps as pivotal decisions (e.g., tool selection, query formulation) that determine trajectory success. Starting from *failed policy trajectories*, we use a PRM to efficiently identify candidate critical steps where the policy errs but expert alternatives show promise. We verify these candidates through branch rollouts: replacing the policy's action with an expert alternative and continuing with the policy itself until completion. Only verified successful branches become preference pairs, contrasting the successful alternative against the original failed action. This yields verifiable, fine-grained preference data that achieves semi-on-policy coverage, offline training efficiency, and outcome-verified supervision.

We evaluate on GAIA (Mialon et al., 2023) and XBench-DeepSearch (Chen et al., 2025) using CK-Pro-8B (Fang et al., 2025). Our contributions are threefold: **(1)** We identify limitations of existing approaches and propose CSO, which focuses on verified critical steps where alternative actions demonstrably flip task outcomes. **(2)** CSO achieves 37% and 26% relative gains over SFT baseline on GAIA-Text-103 and XBench-DeepSearch, enabling an 8B model to match GPT-4.1, while requiring supervision at only 16% of trajectory steps and outperforming all baseline methods. **(3)** We demonstrate that combining PRM identification with outcome verification effectively pinpoints critical decision steps in agent execution, achieving superior performance by focusing learning on these pivotal branches that determine task success.

2 Related Work

2.1 Preference-based Post-training of Agentic Models

Building on base LLMs, recent research has explored various strategies to bolster agentic capabilities. For instance, Li et al. (2025b) integrate a Deep Web Explorer into a think-search-draft loop, utilizing DPO with human feedback for complex report generation. Similarly, Wu et al. (2025b) construct localized preference pairs from debugging traces to apply targeted preference optimization. Our approach is most closely related to Goldie et al. (2025), which synthesizes step-wise reasoning data and employs preference-based RL to fine-tune the model. We extend this framework to agent domains by introducing a method to identify critical steps, thereby improving computational efficiency and

reducing complexity.

2.2 Entropy-enhanced Algorithms for RL

Recent findings in RLVR, particularly Wang et al. (2025c), demonstrate that only a small fraction of high-entropy tokens fundamentally drive effective reinforcement learning for reasoning tasks. Similarly, Cheng et al. (2025) also found that assigning advantages in accordance with entropy leads to better reasoning performances. In terms of the long-horizon agent domain, Wang et al. (2025a); Xu et al. (2025) further proposed entropy-based algorithms to enhance agents' training. This suggests that selective supervision on entropy, which corresponds to critical decision points, may be more effective than uniform or dense supervision. Inspired by this principle, our work identifies critical branching points in agent trajectories and applies verified, fine-grained supervision selectively at these pivotal decisions, combining the reliability of outcome-based verification with the precision of step-level supervision.

3 Methodology

We formalize the agentic decision-making process as ReAct trajectories (§3.1) and introduce our **Critical Step Optimization (CSO)** framework. Our key idea is to identify verified critical steps in failed policy trajectories where alternative actions can flip task outcomes, and construct fine-grained preference pairs from these verified critical steps for targeted DPO training (§3.2). This process can be applied iteratively to progressively refine the policy (§3.3). Figure 2 illustrates the complete CSO pipeline.

3.1 Preliminary

Problem Formulation We consider an agent operating in an interactive environment $\mathcal{E}$ to solve a task specified by query q. Following the ReAct paradigm (Yao et al., 2023), the agent generates a trajectory

$$\tau = (s_1, a_1, o_1, s_2, a_2, o_2, \ldots, s_T, a_T, o_T),$$

where at each step t: s_t represents the *state*, consisting of the task query and interaction history $s_t = (q, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$; $a_t \sim \pi_\theta(\cdot|s_t)$ denotes the action sampled from the policy, which may include both reasoning and tool invocation; $o_t = \mathcal{E}(s_t, a_t)$ is the *observation* returned by the environment. The trajectory terminates at step T with outcome $y \in \{0, 1\}$, where $y = 1$ indicates success.

Supervised Fine-Tuning. We assume access to a base language model fine-tuned on high-quality agent trajectories. Given a dataset $\mathcal{D}_{\text{SFT}} = \{(q^{(i)}, \tau^{(i)})\}$ of successful trajectories:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(q,\tau)\sim\mathcal{D}_{\text{SFT}}}\left[\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)\right] \quad (1)$$

This produces a policy π_θ that can execute agent tasks but may exhibit systematic failures at critical decision points. We refine this policy through preference learning targeting these critical steps.

3.2 Verified Critical Step Collection

Unlike trajectory-level DPO that contrasts entire successful and failed trajectories, or step-level methods that provide estimated but often inaccurate rewards at every step, our approach focuses on *verified critical steps*, meaning steps where taking an alternative action can demonstrably flip the task outcome from failure to success.

Collecting Failed Policy Trajectories. We deploy the current policy π_θ on tasks $\{q^{(i)}\}$ and retain only failed trajectories $\mathcal{T}_{\text{fail}} = \{\tau^{(i)} : y^{(i)} = 0\}$. By starting from the policy's own failures rather than expert demonstrations, we ensure training data remains within the policy's reachable distribution and directly addresses its weaknesses.

Selecting Candidate Critical Steps with PRM Scoring For each failed trajectory $\tau \in \mathcal{T}_{\text{fail}}$, we identify critical steps through a combined process of expert exploration and PRM evaluation. At each step t of the failed policy trajectory:

1. **Expert Alternative Generation:** Sample k alternative actions $\{a'_{t,1}, \ldots, a'_{t,k}\}$ from a stronger expert model π_{expert} conditioned on state s_t.
2. **PRM Scoring:** Use a Process Reward Model (PRM) to score both the policy's original action a_t and each expert alternative $a'_{t,j}$, obtaining scores r_t^{policy} and $\{r_{t,j}^{\text{expert}}\}_{j=1}^{k}$. The PRM produces scores in the range of $[0, 1]$. In practice, we implement PRM by prompting strong closed-source models such as **Claude 3.7 Sonnet** with rubric-based evaluation (see Appendix A for details).
3. **Candidate Critical Step Selection:** Step t is identified as a candidate critical step if the policy

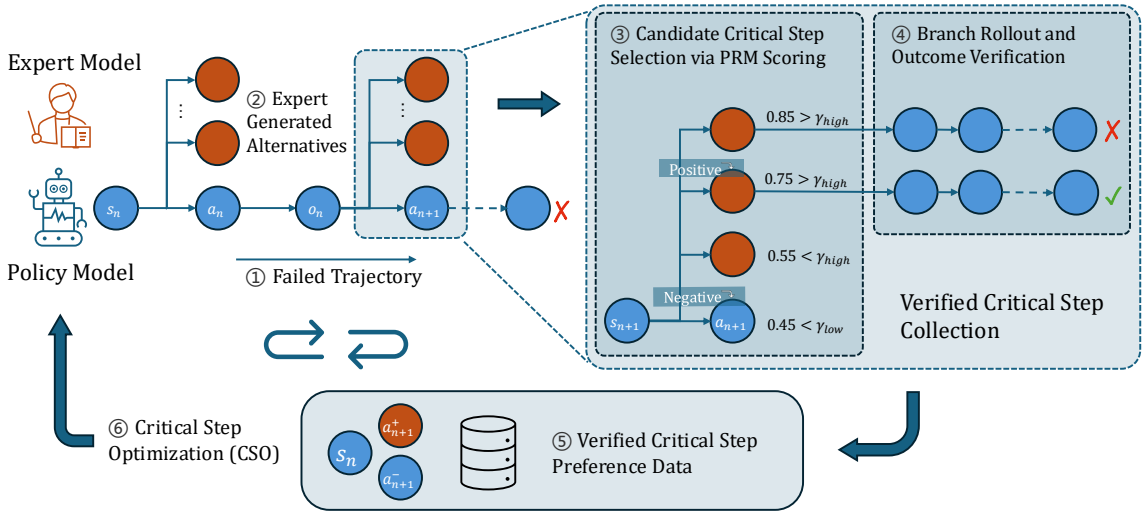

Figure 2: Overview of Critical Step Optimization. (1) The policy model generates trajectories on training queries and we collect failed trajectories; (2) an expert teacher model generates alternative actions at each step; (3) a Process Reward Model identifies candidate critical steps where policy actions have low quality but expert alternatives have high quality; (4) we perform branch rollouts with the policy model and verify task outcomes; (5) we construct preference pairs from verified critical steps and the original failed policy steps; (6) we train the policy via DPO on the verified critical step preference data.

action has low quality while at least one expert alternative has high quality:

$$t \in \mathcal{C}_\tau \iff r_t^{\text{policy}} < \gamma_{\text{low}} \text{ and } \max_j r_{t,j}^{\text{expert}} > \gamma_{\text{high}} \quad (2)$$

where γ_{low} and γ_{high} are quality thresholds.

This PRM scoring mechanism efficiently identifies potential critical steps where the policy makes errors that expert alternatives can correct, without requiring exhaustive Monte Carlo rollouts as in IPR (Xiong et al., 2024). To avoid noise arising from potentially inaccurate PRM estimates, we further verify each identified candidate critical step by checking whether expert alternatives at these steps actually result in successful task completion.

Branch Rollout and Verification. For each candidate critical step $t \in \mathcal{C}_\tau$, we verify which expert alternatives lead to successful outcomes:

1. **Policy Rollout from Alternative Branch:** For each high-scoring expert alternative $a'_{t,j}$ (where $r_{t,j}^{\text{expert}} > \gamma_{\text{high}}$), we construct a new trajectory τ'_j by replacing the original action a_t with the expert alternative $a'_{t,j}$, then continuing rollout with the policy model π_θ until termination. Crucially, all subsequent steps $a'_\ell \sim \pi_\theta(\cdot|s'_\ell)$ for $\ell > t$ are executed by the policy itself, ensuring successful branches remain within its capability.
2. **Outcome Verification:** Evaluate each branched trajectory τ'_j using ground-truth outcome y'_j. Each successful branched trajectory where $y'_j = 1$ corresponds to a *verified critical step* at t, with its alternative action $a'_{t,j}$ as the successful alternative.

This ensures preference data is grounded in actual task success rather than noisy reward estimates. By validating through the policy's own execution, we avoid distribution mismatch where training targets are unreachable by the current policy.

Preference Dataset Construction. For each verified critical step at t, we construct a preference pair (s_t, a_t^+, a_t^-) where a_t^+ is the successful alternative action, a_t^- is the original failed action, and s_t is the shared state context. The final dataset is $\mathcal{D}_{\text{pref}} = \{(s_t, a_t^+, a_t^-)\}$.

3.3 CSO Training

CSO Training Objective We train the policy using Direct Preference Optimization (Rafailov et al., 2023):

$$\mathcal{L}_{\text{CSO}}(\theta) = -\mathbb{E}_{(s_t,a_t^+,a_t^-)\sim\mathcal{D}_{\text{pref}}}\Big[\log\sigma\Big(\beta\log\frac{\pi_\theta(a_t^+|s_t)}{\pi_{\text{ref}}(a_t^+|s_t)} - \beta\log\frac{\pi_\theta(a_t^-|s_t)}{\pi_{\text{ref}}(a_t^-|s_t)}\Big)\Big] \quad (3)$$

where π_{ref} is the reference policy, β is the KL penalty coefficient, and σ is the logistic function.

Unlike trajectory-level DPO that applies signals uniformly across all steps, CSO concentrates learning on verified critical decision points. Unlike step-level methods relying on estimated rewards, CSO's supervision is grounded in verified outcomes, eliminating reward noise.

Iterative Online Refinement. Our framework extends to iterative online training. After each round, we deploy $\pi_{\theta_{\text{new}}}$ to collect fresh failed trajectories and repeat the process:

$$\pi_{\theta_0} \xrightarrow{\mathcal{D}_{\text{pref}}^{(0)}} \pi_{\theta_1} \xrightarrow{\mathcal{D}_{\text{pref}}^{(1)}} \pi_{\theta_2} \xrightarrow{\mathcal{D}_{\text{pref}}^{(2)}} \cdots \tag{4}$$

At iteration i, we set $\pi_{\text{ref}} = \pi_{\theta_{i-1}}$ for stable training. As the policy improves, critical steps may shift toward harder decision points, enabling progressive refinement. This combines offline DPO stability with online policy adaptivity, avoiding full RL overhead while maintaining semi-on-policy coverage.

4 Experiments

4.1 Experimental Settings

Policy Model and Framework. We use **CK-Pro-8B** (Fang et al., 2025) as our policy model, an 8B-parameter agent model obtained by supervised fine-tuning (SFT) on Qwen3-8B (Yang et al., 2025) base model. All agent interactions are executed through the **Cognitive Kernel Pro** framework, which is a fully open-source and free multi-module agent framework. It features a main agent responsible for orchestrating specialized sub-agents dedicated to web navigation, file handling, and tool invocation, providing a standardized and extensible environment for agent usage. We extend the Cognitive Kernel Pro framework to support two critical features for our experiments: (1) execution of Process Reward Model (PRM) scoring directly within agent steps, and (2) seamless continuation of agent rollouts from any given intermediate state. We leverage the **LlamaFactory** (Zheng et al., 2024) framework for all post-training.

Training Data and Expert Model. Our DPO preference data is constructed starting from the SFT training data of CK-Pro-8B, which contains 47K task-trajectory pairs covering diverse reasoning and tool-use scenarios. We deploy the policy model on these tasks to collect failed trajectories, then use **Claude-3.7-Sonnet** as the expert model to generate high-quality alternative actions.

Hyperparameters. For DPO training, we set the KL penalty coefficient $\beta = 0.5$. For critical step identification, we sample $K = 5$ alternative candidates at each potential branching point. We conduct iterative training for up to 2 rounds, where each round collects fresh failed trajectories from the updated policy. PRM scoring uses **Claude-3.7-Sonnet** with rubric-based evaluation (see Appendix A). For PRM thresholds, we set $\gamma_{\text{high}} = 0.65$ and $\gamma_{\text{low}} = 0.45$ to identify candidate critical steps. All experimental results are reported based on three independent runs to mitigate variability due to network conditions and external factors. As web search is involved in several tasks, some results may exhibit minor differences from the original papers.

4.2 Benchmarks and Baselines

Benchmarks. We evaluate on two challenging agent benchmarks: **GAIA-Text-103** (Mialon et al., 2023): As our model is text-only, we follow WebThinker (Li et al., 2025b) and report the text-only subset of the GAIA benchmark. This subset contains 103 questions spanning three difficulty levels (L1, L2, L3) requiring multi-step reasoning and tool orchestration. We report this subset as our models are pure text-based agents. **XBench-DeepSearch** (Chen et al., 2025): A complex information retrieval and reasoning benchmark requiring deep search strategies and evidence synthesis across multiple web sources. We choose 2505 version which contains 100 complex tasks. For all benchmarks, we follow the evaluation protocol of WebThinker and CK-Pro-8B, using an LLM to determine whether each output is correct, referencing the gold answer as ground truth.

Baselines We compare against several categories of methods: **Proprietary Models**: GPT-4.1 and Claude-3.7-Sonnet as strong closed-source baselines. **Open-Source Models**: Qwen3-8B (base model) and CK-Pro-8B (SFT baseline). **Post-Training Methods**: We implement several post-training baselines within the **Cognitive Kernel Pro** framework to ensure fair comparison. **Exploration-based Trajectory Optimization (ETO)** (Song et al., 2024) constructs preference pairs by using expert-generated successful trajectories as positive examples and policy-generated failed trajectories as negative examples, applying trajectory-level DPO. **Rejection Sampling Fine-Tuning (RFT)** (Zhang et al., 2025) collects successful tra-

Model / Method	GAIA-Text-103				XBench-DeepSearch2505
	L1 (%)	L2 (%)	L3 (%)	All (%)	Score
Proprietary Models					
GPT-4.1	56.4	44.2	16.7	45.6	27.0
Claude-3.7-Sonnet	76.9	57.7	33.3	62.1	41.0
Open-Source Baselines					
Qwen3-8B	35.9	13.5	0.0	20.4	7.0
CK-Pro-8B (SFT)	46.2	34.6	8.3	35.9	23.0
Post-Training Methods on CK-Pro-8B					
+ ETO	51.2	36.5	8.3	38.9	22.0
+ RFT	51.2	28.8	8.3	34.9	20.0
+ Step-DPO	53.3	34.6	8.3	38.9	25.0
+ IPR	56.4	42.3	16.7	44.6	24.0
+ CSO (Ours)	**61.5**	**48.1**	**16.7**	**49.5**	**29.0**

Table 1: Performance comparison on GAIA-Text-103 and XBench-DeepSearch benchmarks. Our Critical Step Optimization achieves the best performance among all post-training methods applied to CK-Pro-8B, demonstrating the effectiveness of selective supervision at verified critical steps.

jectories generated by the policy model itself and performs supervised fine-tuning on these successful cases. **Step-wise DPO** (Choudhury, 2025) applies dense step-level preference learning at every step in trajectories using PRM scoring; notably, we use the same PRM implementation (Claude-3.7-Sonnet with rubric-based evaluation) as our method to control for PRM quality. **Iterative Process Refinement (IPR)** (Xiong et al., 2024) constructs step-level rewards by identifying steps in outcome-verified successful trajectories as positive examples and corresponding steps in failed policy trajectories as negative examples, applying step-level DPO with these outcome-grounded signals.

4.3 Main Results

CSO achieves strong performance matching proprietary models. Table 1 presents the performance comparison across all methods. Our Critical Step Optimization achieves 49.5% overall accuracy on GAIA-Text-103, a 37% relative improvement over the SFT baseline. Notably, CSO enables the open-source 8B model CK-Pro-8B to match the performance of GPT-4.1. CSO achieves consistent gains across all difficulty levels and substantially outperforms all post-training baselines by at least +5.0 points on both GAIA and XBench-DeepSearch.

Trajectory-level methods show limited effectiveness. RFT shows no significant change (34.9%) by training on the policy's own successful trajectories. ETO performs better (38.9%, +3.0 points) by contrasting expert successes against policy failures. Both methods suffer from coarse credit assignment by applying outcome-based rewards uniformly across all steps.

Step-level and hybrid methods face accuracy and noise issues. Step-DPO achieves strong performance on simple L1 tasks (+7.1 points) but shows no improvement on harder L2/L3 tasks, revealing PRM accuracy degradation on complex reasoning. IPR improves through outcome verification (44.6% overall) but still suffers from outcome reward contamination. In contrast, CSO achieves +5.0 points over IPR by focusing exclusively on verified critical steps with precise credit assignment.

5 Analysis

5.1 CSO Ablations

Expert Positive Examples with Policy Negative Examples Achieve Best Performance. To understand what types of preference pairs contribute most to learning, we systematically compare three data source configurations: (1) expert successes with expert failures, (2) policy successes with policy failures, and (3) expert successes with policy failures. All configurations start from the same failed policy trajectories and identify the same critical steps, differing only in the source of positive

and negative actions. As shown in Table 2, combining expert successes with policy failures achieves the best performance, substantially outperforming both expert-only and policy-only pairs. This suggests that the most effective learning signal comes from contrasting expert demonstrations with the policy's own failure modes at critical junctures, enabling the model to learn from high-quality behaviors while recognizing its specific weaknesses.

Table 2: Comparison of preference data sources for CSO.

Data Source	GAIA-Text
Expert Success + Expert Failure	46.6
Policy Success + Policy Failure	42.7
Expert Success + Policy Failure	**49.5**

Combining PRM Selection and Outcome Verification Ensures Both Performance and Efficiency. We investigate the effectiveness of combining PRM selection with outcome verification. As shown in Table 3, we compare three strategies: (1) outcome verification only, (2) PRM selection only, and (3) combining both. The results demonstrate two key findings. First, outcome verification is critical: methods with verification substantially outperform PRM-only selection and cost only 16% steps. Second, while skipping PRM selection achieves comparable performance, it requires nearly 3× more preference pairs. This validates our design: PRM selection efficiently narrows candidate critical steps while outcome verification ensures final quality, achieving both high performance and sample efficiency.

Table 3: Ablation on PRM selection and outcome verification strategies. #Samples indicates the number of preference pairs constructed.

Strategy	GAIA-Text (%)	#Samples
PRM + Verification	**49.5**	**671**
w/o PRM	48.5	1,967
w/o Verification	43.6	4,126

Number of Branch Candidates. We investigate the impact of the number of sampled candidates k at each critical step. As shown in Table 4, appropriately increasing k enhances the diversity and representativeness of candidate samples, improving performance from 46.6% at $k = 3$ to 49.6% at $k = 5$ on GAIA-Text. However, increasing k beyond 5 does not lead to further improvement in performance, but substantially increases the verification cost; thus, larger k offers limited practical benefit given the rising computational overhead. These results suggest that $k = 5$ strikes an optimal balance between exploration quality and efficiency.

Table 4: Impact of the number of branch candidates k at each critical step.

k (Branches)	GAIA-Text (%)	XBench (%)
$k = 3$	46.6	26.0
$k = 5$	**49.6**	**29.0**
$k = 7$	49.6	28.0

5.2 Impact of PRM Quality and Usage

We analyze how PRM quality and usage affect final performance. Specifically, we compare two PRM sources (Claude-3.7-Sonnet and GPT-4.1) and two usage paradigms: (1) CSO, which leverages the PRM for candidate selection combined with outcome verification, and (2) step-level Best-of-N (BoN), which uses PRM to directly select actions from sampled candidates. All experiments are conducted on CK-Pro-8B and evaluated on GAIA-Text-103-L1. In both settings, the number of candidates k per step is fixed at 5 to ensure a controlled comparison.

Table 5: Impact of PRM quality and usage on performance. CSO uses PRM for candidate selection with outcome verification, while BoN uses PRM to guide search by selecting from policy candidates.

PRM Source	CSO	Step-level BoN
Claude-3.7-Sonnet	**61.5**	56.2
GPT-4.1	53.3	48.7

The results indicate two key observations. First, the strength of the underlying foundation model is critical for PRM quality: Claude-3.7-Sonnet consistently yields higher performance than GPT-4.1 as a PRM, suggesting that models with stronger downstream task performance produce more reliable process rewards. Second, the CSO paradigm integrating PRM selection with explicit outcome verification achieves superior results relative to PRM-guided search alone. Notably, for the same PRM, CSO markedly outperforms step-BoN, highlighting CSO's ability to mitigate PRM noise through

outcome-based validation, whereas BoN is fully dependent on potentially noisy PRM-driven judgments. These findings validate our approach of using the PRM primarily for efficient candidate identification, subsequently filtered by robust outcome verification, rather than as a direct arbiter for action selection.

5.3 Analysis On Iterative Online Refinement

We compare CSO against ETO (Song et al., 2024) and IPR (Xiong et al., 2024) across four online training rounds on CK-Pro-8B, evaluated on GAIA-Text-103-L1. As shown in Figure 3, the three methods exhibit distinctly different behaviors. ETO initially improves but then degrades significantly, falling below the SFT baseline by Round 3 due to trajectory-level supervision uniformly penalizing all steps in failed trajectories, causing the policy to unlearn correct behaviors. IPR achieves more stable improvement, reaching 56.4% at Round 2-3, benefiting from Monte Carlo step-level rewards. However, IPR still propagates trajectory-level outcome signals to all steps, limiting further gains. In contrast, CSO achieves 61.5% by Round 2 and maintains this through Round 3, consistently outperforming both baselines. By focusing exclusively on verified critical steps, CSO avoids credit assignment noise and enables more effective online improvement.

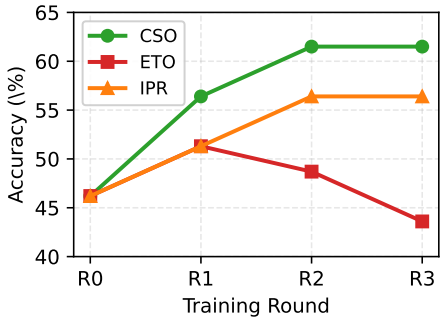

Figure 3: Performance across online training iterations. All methods trained on CK-Pro-8B, evaluated on GAIA-Text-103-L1.

5.4 Categorizing Critical Steps

We manually analyze a subset of PRM-identified critical steps and categorize them by error type. As shown in Figure 4, the identified critical steps distribute across several categories: Tool Invocation errors (26.1%) represent the largest category, including incorrect tool selection or suboptimal query formulation for search and retrieval operations. Reasoning Errors (25.1%) involve logical mistakes or incorrect calculations during task execution. Other Errors (24.1%) include miscellaneous issues such as parsing errors or edge case handling. Task Understanding errors (13.0%) stem from misinterpreting task requirements, while Information Extraction errors (11.7%) occur when the agent locates relevant information but fails to extract it correctly. This distribution demonstrates that CSO effectively identifies diverse types of critical decision points where the policy makes pivotal errors, validating that our method targets semantically meaningful steps rather than arbitrary positions in trajectories.

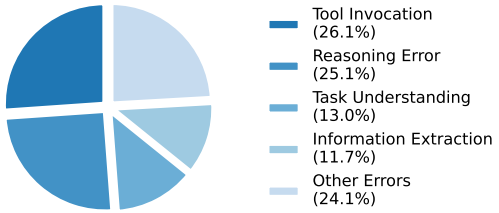

Figure 4: Distribution of critical step error types identified by CSO. Tool invocation and reasoning errors constitute the majority of critical decision points.

6 Conclusion

We introduced Critical Step Optimization (CSO), a post-training approach that focuses preference learning on verified critical steps where alternative actions demonstrably flip task outcomes. Inspired by findings that only high-entropy tokens drive effective RL for reasoning, we demonstrate that agent trajectories similarly contain a sparse subset of critical steps that determine success. Our method combines PRM selection with outcome verification to construct fine-grained preference data without trajectory-level coarseness or step-level estimation noise. Experiments show that CSO achieves 37% and 26% relative improvements over SFT on GAIA-Text-103 and XBench-DeepSearch, enabling an open-source 8B model to match GPT-4.1 while requiring supervision at only 16% of steps. CSO substantially outperforms trajectory-level, dense step-level, and hybrid methods, demonstrating that selective verified supervision provides an efficient framework for agent post-training.

Limitations

Our approach has two main limitations that present opportunities for future work. First, outcome verification requires executing trajectories to completion to confirm correctness, which can be time-consuming on complex tasks. While this ensures high-quality supervision in offline settings, applying CSO to online RL scenarios would require addressing this efficiency bottleneck. Potential solutions such as early stopping heuristics or parallelized execution could make CSO more practical for online learning, likely yielding strong performance gains. Second, our current implementation relies on closed-source models (Claude-3.7-Sonnet) as the PRM, which cannot be jointly optimized with the policy model. As open-source models continue to improve and approach the quality of top commercial systems, jointly training the PRM alongside the policy could further enhance performance. We leave these directions to future work.

Ethics Statement

This work adheres to ethical research practices and open science principles. All data, models, and frameworks used in our experiments are sourced from the open-source community and comply with their respective licenses. The only paid service employed is the Google Search API for web search functionality, which is used in accordance with its terms of service. Importantly, our data annotation pipeline relies entirely on automated PRM scoring and outcome verification, which requiring no human labor for preference data construction. This eliminates concerns related to annotator exploitation or bias injection through human judgment. Our work aims to democratize access to high-quality agent systems by demonstrating effective post-training techniques for open-source models. We used ChatGPT to assist with grammar checking in this manuscript.

References

2025. Deepseek-v3.2: Pushing the frontier of open large language models.

Anthropic. 2025. Claude-sonnet-4-5-system-card.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xuanzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. 2025. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. 2025. Reasoning with exploration: An entropy perspective.

Sanjiban Choudhury. 2025. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Tianqing Fang, Zhisong Zhang, Xiaoyang Wang, Rui Wang, Can Qin, Yuxuan Wan, Jun-Yu Ma, Ce Zhang, Jiaqi Chen, Xiyun Li, Hongming Zhang, Haitao Mi, and Dong Yu. 2025. Cognitive kernel-pro: A framework for deep research agents and agent foundation models training.

Anna Goldie, Azalia Mirhoseini, Hao Zhou, Irene Cai, and Christopher D. Manning. 2025. Synthetic data generation & multi-step rl for reasoning & tool use.

Yeonsung Jung, Trilok Padhi, Sina Shaham, Dipika Khullar, Joonhyun Jeong, Ninareh Mehrabi, and Eunho Yang. 2025. Co-evolving agents: Learning from failures as hard negatives.

Kimi-Team. 2025. Kimi k2: Open agentic intelligence.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Websailor: Navigating superhuman reasoning for web agent. *arXiv preprint arXiv:2507.02592*.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025b. Webthinker: Empowering large reasoning models with deep research capability.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants.

MiroMind. 2025. Mirothinker: Pushing the performance boundaries of open-source research agents via model, context, and interactive scaling.

OpenAI. 2024. OpenAI o1: Introducing openai's reasoning model. Online; OpenAI model card and announcement. The o1 model is designed for enhanced complex reasoning tasks.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*. Also available as preprint on arXiv.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025. Webshaper: Agentically data synthesizing via information-seeking formalization.

Jiawei Wang, Jiacai Liu, Yuqian Fu, Yingru Li, Xintao Wang, Yuan Lin, Yu Yue, Lin Zhang, Yang Wang, and Ke Wang. 2025a. Harnessing uncertainty: Entropy-modulated policy gradients for long-horizon llm agents.

Rui Wang, Ce Zhang, Jun-Yu Ma, Jianshu Zhang, Hongru Wang, Yi Chen, Boyang Xue, Tianqing Fang, Zhisong Zhang, Hongming Zhang, Haitao Mi, Dong Yu, and Kam-Fai Wong. 2025b. Explore to evolve: Scaling evolved aggregation logic via proactive online exploration for deep research agents.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. 2025c. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*.

Jie Wu, Haoling Li, Xin Zhang, Xiao Liu, Yangyu Huang, Jianwen Luo, Yizhen Zhang, Zuchao Li, Ruihang Chu, Yujiu Yang, and Scarlett Li. 2025b. Teaching your models to understand code via focal preference alignment. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14003–14023, Suzhou, China. Association for Computational Linguistics.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Wujiang Xu, Wentian Zhao, Zhenting Wang, Yu-Jhe Li, Can Jin, Mingyu Jin, Kai Mei, Kun Wan, and Dimitris N. Metaxas. 2025. Epo: Entropy-regularized policy optimization for llm agents reinforcement learning.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.

Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. 2025. Cumulative reasoning with large language models.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.

A PRM Prompt

As described in Section 3.2, we use a Process Reward Model (PRM) to evaluate the quality of both policy actions and expert alternatives at each step of failed trajectories. In our implementation, we employ strong closed-source models (specifically Claude 3.7 Sonnet) with carefully designed rubric-based evaluation prompts to serve as the PRM. The prompt instructs the model to assess action quality across multiple dimensions including code correctness, task relevance, logical progression, information utilization, and thought quality. The PRM produces scores in the range $[0, 1]$, which are used to identify candidate critical steps where policy actions have low quality ($r_t^{\text{policy}} < \gamma_{\text{low}}$) but expert alternatives have high quality ($\max_j r_{t,j}^{\text{expert}} > \gamma_{\text{high}}$). Figure 5 shows the complete prompt used for PRM evaluation.

Execution Process Reward Model (PRM) Prompt
You are a Process Reward Model (PRM) responsible for critically evaluating the quality of agent actions during task execution. Your role is to rigorously assess whether a proposed action is likely to make meaningful progress toward completing the given task.
IMPORTANT: You are a STRICT evaluator. Most actions should score between 0.4-0.8. Only truly exceptional actions deserve scores above 0.85. Be critical and look for flaws.
CRITICAL CAUTION ON DATA RELIABILITY: Answers or fields retrieved from Hugging Face GAIA datasets (e.g., dataset-provided "answer"/metadata fields) are OFTEN WRONG and MUST NOT be treated as ground truth. Penalize any action that blindly copies, trusts, or cites GAIA dataset fields without independent verification from reliable sources or prior validated state/history. Actions should explicitly verify claims and cross-check sources; lack of verification is a scoring liability.
Evaluation Rubric:
1. **Code Correctness & Detail** (35%) - CRITICAL: **1.0:** Code is flawless with perfect syntax, logic, edge case handling, and data type management. **0.85:** Code is correct but could be more robust (missing 1-2 minor edge cases). **0.7:** Code is mostly correct but has 2-3 potential issues (type mismatches, off-by-one errors, missing error handling). **0.5:** Code has notable bugs that will likely cause partial failures. **0.3:** Code has major logical errors or will fail in most cases. **0.0:** Code is fundamentally broken or will definitely fail. **Critical checks for complex code:** Variable initialization and scope, loop boundaries and termination conditions, list/array indexing (off-by-one errors are common!), type compatibility (string vs int, list vs dict), error handling and edge cases, function call signatures and return values, import statements and dependencies.
2. **Task Relevance** (25%): **1.0:** Action perfectly addresses the exact task requirement with optimal approach. **0.75:** Action addresses the task well but approach is not optimal. **0.5:** Action is relevant but takes an indirect or inefficient path. **0.25:** Action has minimal relevance or addresses the wrong aspect. **0.0:** Action is completely irrelevant or counterproductive.
3. **Logical Progression** (20%): **1.0:** Action perfectly builds on previous steps, avoiding redundancy and utilizing all prior results. **0.75:** Action follows logically but may repeat some work unnecessarily. **0.5:** Action makes sense but shows gaps in utilizing previous progress. **0.25:** Action shows weak logical connection to previous steps. **0.0:** Action contradicts or ignores previous progress.
4. **Information Utilization** (15%): **1.0:** Leverages ALL relevant information from state, history, and task description. **0.75:** Uses most key information effectively. **0.5:** Uses some information but misses important details from state or history. **0.25:** Mostly ignores available information. **0.0:** Completely fails to utilize or actively misuses available information. NOTE: Reliance on unverified dataset-provided answers (e.g., GAIA fields) counts as misuse unless independently verified.
5. **Thought Quality & Planning** (5%): **1.0:** Thought shows deep understanding with clear, detailed reasoning. **0.75:** Thought is clear and logical. **0.5:** Thought is vague or shows partial understanding. **0.25:** Thought is unclear or shows misunderstanding. **0.0:** Thought is missing or completely wrong.
Strict Scoring Guidelines: **0.9-1.0** (Exceptional - RARE): Near-perfect code with excellent thought, optimal approach, perfect logic. **0.8-0.89** (Excellent): Very good code with minor room for improvement, strong thought and logic. **0.7-0.79** (Good): Solid code with 1-2 fixable issues, reasonable approach. **0.6-0.69** (Acceptable): Code works but has several issues or suboptimal approach. **0.5-0.59** (Mediocre): Code has notable problems, weak logic or poor information use. **0.4-0.49** (Poor): Significant code issues or wrong approach, likely to fail partially. **0.3-0.39** (Bad): Major flaws in code or logic, will likely fail. **0.0-0.29** (Failure): Fundamentally broken or irrelevant.
Common Code Pitfalls to Penalize: (1) Off-by-one errors in loops and indexing (reduce by 0.15-0.25). (2) Type mismatches (string concatenation with ints, etc.) (-0.1-0.2). (3) Missing imports for used libraries (-0.1-0.15). (4) Variable name typos or inconsistent naming (-0.05-0.15). (5) Edge case failures (empty list, None values, zero division) (-0.1-0.2). (6) Incorrect function signatures for tools/sub-agents (-0.2-0.3). (7) Missing error handling in critical sections (-0.05-0.15). (8) Inefficient algorithms when better approaches exist (-0.05-0.15). (9) Redundant work that ignores previous results (-0.1-0.2). (10) Poor data structure choice (-0.05-0.1). (11) Unverified or incorrect source usage (-0.2-0.4): blindly trusting dataset-provided fields without verification; fabricating or misattributing sources; failing to cross-check with retrieved content or authoritative references. (12) Failure to identify the error source when a mistake occurs (-0.1-0.2): reasoning should pinpoint whether the issue came from dataset fields, parsing, tool output, stale state, or coding logic.
Response Format: Provide reasoning (3-4 sentences providing specific, detailed assessment. Mention specific code issues if any, explain scoring decisions, reference rubric criteria. If a mistake exists, explicitly identify the ERROR SOURCE, e.g., "relied on GAIA dataset field likely incorrect", "misparsed web content", "stale state variable", "wrong tool signature") and a score (single decimal 0.0-1.0).

Prompt 5: Process Reward Model prompt for evaluating the quality of agent actions (code) during task execution. The PRM critically assesses code correctness, task relevance, logical progression, information utilization, and thought quality, with special attention to data reliability issues.