PLUG-AND-PLAY VERIFIER FOR DECOUPLING PROCESS REWARD IN STABILIZING LONG-HORIZON AGENTIC REASONING

Anonymous authors
Paper under double-blind review

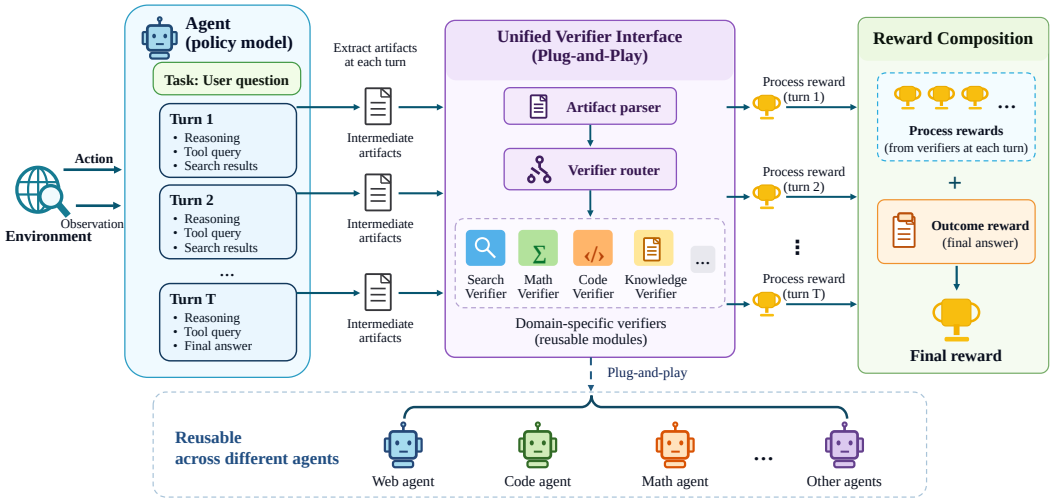

Figure 1: **Overview of the plug-and-play verifier.** *Left:* artifacts are extracted per turn. *Middle:* a unified interface routes them to domain-specific verifiers, which return explicit and decoupled process rewards. *Right:* Rewards are combined with the final outcome reward. *Bottom:* verifiers interact with RL post-training through the interface and can be used independently across agents.

ABSTRACT

Stabilizing long-horizon reasoning requires intelligent agents to sustain useful progress across successive reasoning steps and tool interactions. Such agents are typically trained with reinforcement learning (RL), where actor-critic RL uses a learnable critic network to estimate step advantages, while critic-free methods such as GRPO eliminate critic models and rely mostly on the group variants of sparse outcome-only rewards. Both reward types leave partially useful failed trajectories either implicit or indistinguishable from entirely unproductive ones, and as reasoning horizons grow, it becomes difficult to maintain consistent reasoning traces. In this work, we propose a *plug-and-play verifier* framework for stabilizing long-horizon reasoning through explicit verification of intermediate reasoning artifacts. Specifically, the framework decouples process verification through a unified interface that extracts intermediate reasoning artifacts from agent running trajectories, evaluates them using domain-specific verifiers, and aggregates the verification grades as bounded process reward. Then, we combine our dense verified process rewards and the sparse final outcome through a two-stage post-training schedule. We showed that with plug-and-play verifiers, we further stabilize the reasoning steps as last-mile self-reflection and self-consistency. Typical evaluation on HotpotQA using Qwen3.5-4B achieves a superior accuracy (60.18%), outperforming the standard GRPO baseline (57.73%). Our results suggest that plug-and-play verifiers provide a modular approach to learn more stable and reliable reasoning behaviors over long horizons.

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.

1 INTRODUCTION

Large language models are increasingly deployed as agents that solve tasks by interleaving reasoning with actions in external environments, following the reasoning, acting, and observing cycle proposed by ReAct (Yao et al., 2022). Equipped with tools and environment feedback, these agents tackle information seeking, software engineering, and web navigation through sequences of interdependent decisions (Schick et al., 2023; Yang et al., 2024; Zhou et al., 2024; Zheng et al., 2025). Success on these tasks requires effective agentic reasoning, where agents must interpret newly acquired evidence, revise their plans, and carry stable intermediate results forward to guide subsequent actions (Shinn et al., 2023; Zhou et al., 2023; Zhu et al., 2026). However, sustaining this capability becomes increasingly challenging as the interaction horizon grows (Da et al., 2025; Gandhi et al., 2026). For example, a hallucinated claim, calculation error, or misinterpreted tool result can misdirect subsequent actions as later decisions build on earlier reasoning and observations (Yao et al., 2022; Yuan et al., 2026). Evaluations across interactive environments, software engineering, and general assistant tasks further highlight persistent difficulties in reliably completing multi-step tasks (Liu et al., 2024; Jimenez et al., 2024). These difficulties are exacerbated in multi-agent interactions, where an agent's erroneous outputs are passed to others and incorporated into reasoning, allowing errors to propagate and compound across agents (Du et al., 2023; Cui et al., 2026). Therefore, stabilizing long-horizon agentic reasoning is emergent (Xu et al., 2026).

Reinforcement learning provides an effective approach to improving agentic reasoning by allowing models to explore alternative strategies and learn from verifiable outcomes without requiring human annotations for every intermediate step (Wang et al., 2025b; Guo et al., 2025; Jin et al., 2025; Feng et al., 2026a). Both actor-critic methods and critic-free methods such as GRPO rely on the supplied rewards to guide policy optimization (Schulman et al., 2017; Shao et al., 2024). However, rewards are usually implicit and do not distinguish useful intermediate progress in failed trajectories (Yang et al., 2026; Zhang et al., 2025), motivating the necessity of explicit and distinguishable process supervision for long-horizon agentic tasks (Yang et al., 2026; Zhang et al., 2025; Yuan et al., 2026)

In this work, we observe that process rewards can be obtained by checking intermediate artifacts against task conditions and evidence independently of reward composition and policy optimization. Specifically, assessing whether an artifact satisfies a verification criterion does not require deciding how the score is combined with outcome rewards or used to update the policy. Conversely, downstream training requires the verification scores and their source-step attribution from the verifier. This independence allows us to decouple process verification from reward composition and policy optimization. Therefore, we implement process verification in external modules that interact with training through an interface that passes verification scores and their source-step attribution. Moreover, agents in different domains produce different intermediate artifacts. To support agents without adapting the training procedure to each verifier, we propose a unified plug-and-play verification interface. As illustrated in Figure 1, every verification module extracts intermediate artifacts, verifies them, and returns bounded process rewards attributed to their source interaction steps. Since downstream reward composition and policy optimization depend on each verifier only through these outputs, verification modules for different agents can be introduced or replaced in a plug-and-play manner. Inspired by the above, we propose a *plug-and-play verifier* framework for stabilizing long-horizon agentic reasoning through explicit verification of intermediate artifacts. Specifically, our framework makes the following contributions:

- Process rewards by verifying intermediate artifacts against task conditions and environment evidence, providing complementary supervision beyond outcome rewards.
- A unified interface that decouples process verification from reward composition and policy optimization, enabling plug-and-play integration of verifiers.

To evaluate it, we conduct experiments using Qwen3.5-4B and Qwen3.5-9B on the HotpotQA validation split, LiveCodeBench, MATH-Vision, and AIME 2025, covering multi-hop question answering, code generation, multimodal reasoning, and mathematical reasoning. Our experiments show that incorporating a plug-and-play verifier can improve the performance and stabilize the reasoning traces. Our ablation studies further examine verification mechanisms and support the contributions of reward composition. Our results suggest that explicit process verification can complement outcome rewards to improve agentic reasoning.

2 Related Work

Long-horizon reasoning and multi-agent interaction. Long-horizon agentic reasoning requires agents to make successive decisions based on previously generated information and observations from tools or environments. Early approaches support multi-step reasoning through explicit intermediate steps and exploration of alternative reasoning paths (Wei et al., 2022; Yao et al., 2023). Tool-augmented approaches connect reasoning with external information and executable actions, enabling agents to interact with environments and perform tasks (Yao et al., 2022; Schick et al., 2023; Yang et al., 2024). Feedback and episodic memory can guide agents to revise their strategies across successive attempts (Shinn et al., 2023). Multi-agent approaches extend these capabilities through role-specific workflows and iterative debate (Hong et al., 2024; Du et al., 2023). However, evaluations on WebArena and AgentBench reveal persistent difficulties in long-term planning, decision-making, and reliable task completion (Zhou et al., 2024; Liu et al., 2024). Analyses of multi-agent systems identify failures in coordination and task verification (Cemri et al., 2026). Recent analysis of search agents further identifies failures in evidence propagation (Sang et al., 2026).

Agentic reinforcement learning. Reinforcement learning enables agents to learn reasoning and tool-use strategies from task feedback. Search-R1 and ReTool incorporate search and code execution into multi-turn trajectories and optimize interaction policies using outcome rewards (Jin et al., 2025; Feng et al., 2026a). Actor-critic methods use learned value functions to estimate advantages; these functions predict returns rather than verify intermediate artifacts (Schulman et al., 2017). Critic-free methods such as GRPO normalize rewards within a group of rollouts, but under outcome-only supervision, each trajectory receives a single outcome reward (Shao et al., 2024). Research on multi-turn agent training addresses training stability and temporal credit assignment (Wang et al., 2026a;b; He et al., 2026; Yang et al., 2026). More recently, GiGPO combines trajectory-level comparisons with step-level advantage estimation over actions taken at recurring environment states (Feng et al., 2026b). IGPO instead derives turn-level rewards from increases in the policy's probability of producing the correct answer and combines them with outcome supervision (Wang et al., 2025a).

Process supervision and verifiable feedback. Process supervision provides dense feedback on intermediate reasoning beyond sparse final outcome correctness. A central question is how to obtain reliable supervision for these intermediate steps. Initial human-in-the-loop methods demonstrate the effectiveness of the supervision in mathematical reasoning (Lightman et al., 2024), while Math-Shepherd and ReST-MCTS derive supervision automatically from rollouts and tree search (Wang et al., 2024b; Zhang et al., 2024). Rewarding Progress further evaluates intermediate steps by their contribution to a prover policy's likelihood of success (Setlur et al., 2025). Alongside these approaches, another line of work grounds feedback in tools, evidence, or explicit verification criteria. CRITIC uses tools to validate and revise outputs at inference time (Gou et al., 2024), while VPR and VPRMs incorporate symbolic or algorithmic checks and deterministic rules into process rewards for intermediate actions and structured reasoning traces (Yuan et al., 2026; Pronesti et al., 2026). For agentic tasks, this raises a further question of how to attribute feedback to individual actions (Gandhi et al., 2026; Chai et al., 2025). STAMP attributes verified evidence to the actions that first exposed it, while ABSeeker constructs step-level supervision from answer-backtracked clues (Xu et al., 2026; Lu et al., 2026).

3 Methods

3.1 Problem Setting and Preliminary

We study long-horizon agentic reasoning tasks. Given a task domain d, let $x \sim \mathcal{D}_d$ denote an input consisting of a task instruction and any available context, including images for multimodal tasks. A policy π_θ, parameterized by θ, produces a solution through successive policy generations and, when applicable, through interactions with an external environment.

Policy and trajectories. At interaction step t, the policy generates an action $a_t \sim \pi_\theta(\cdot \mid h_t)$ conditioned on the interaction history $h_t = (x, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$. An action is a complete model-generated response that may contain reasoning text, a tool call, or a final answer. When a tool is invoked, the environment returns an observation o_t, which is incorporated into the history for the next step. For a given input x, a completed trajectory τ records the sequence of policy-generated

actions and environment observations in one solution attempt:

$$\tau = (a_1, o_1, \ldots, a_{T-1}, o_{T-1}, a_T), \tag{1}$$

where T is the number of interaction steps, each corresponding to one policy generation, and a_T contains the final answer, denoted by $\hat{y}(\tau)$. Observations are omitted when no external interaction occurs. A single policy generation may contain multiple candidate artifacts; verification is performed at the artifact level, while the resulting process credits are attributed to interaction steps.

Outcome and process rewards. Trajectories with the same final outcome reward may differ in whether their intermediate artifacts satisfy task conditions and are supported by recorded environment evidence. To capture this distinction, we use a process reward alongside the outcome reward:

$$E_d(x, \tau) \in [0, 1], \qquad P_d(x, \tau) \in [0, 1]. \tag{2}$$

The outcome reward $E_d(x, \tau)$ evaluates the final answer $\hat{y}(\tau)$ extracted from trajectory τ against the task-specific correctness criteria associated with input x. Evaluation references, such as reference answers or test cases, are available to the evaluator and need not be included in the policy input. Thus, the outcome reward depends on τ through its final answer, while the process reward $P_d(x, \tau)$ aggregates local verification scores for intermediate artifacts in τ. A passing local check establishes only the specific property tested by its verification rule, but it does not establish outcome correctness. For an outcome-process mixing coefficient $w \in [0, 1]$, the trajectory-level mixed reward is

$$R_{d,w}(x, \tau) = wE_d(x, \tau) + (1 - w)P_d(x, \tau). \tag{3}$$

The coefficient w controls the relative contribution of the two reward components: $w = 1$ yields outcome-only supervision, whereas $w = 0$ yields process-only supervision.

3.2 Plug-and-Play Process Verification

Plug-and-play has two complementary meanings in our framework: verification is separable from the policy's reasoning procedure, and the resulting credits are reusable across compatible agents and optimizers through an explicit interface. Local separability defines what is checked; the interface specifies how those checks are incorporated into training.

Decoupled local verification. Verification assesses an artifact from explicitly specified inputs, without evaluating the full reasoning narrative. A domain-specific extractor $\mathcal{C}_d$ identifies m artifacts in (x, τ) and their associated steps. An artifact is an explicit object admitting a check, such as an arithmetic equality, a citation, or a code version associated with an execution record. We define the score by

$$v_j = V_d(c_j; \xi_j) \in [0, 1], \tag{4}$$

where ξ_j contains the task conditions and records required by the check. These may include earlier artifacts or specified submission fields. For fixed rules and evaluation conditions, the score depends on the trajectory only through (c_j, ξ_j); changing other reasoning text or its final correctness label does not change the score. The verifier does not access policy parameters or hidden states, sample policy continuations, or estimate eventual task success.

This separation allows checks to be replayed after rollout completion when their required inputs have been recorded. It does not require artifacts to be independent; their dependencies remain explicit in ξ_j. Nor does it imply the absence of task-specific verification resources. The QA verifier uses source-span grounding and action-certificate matching without annotated supporting facts, the code verifier uses version lineage and developer-test replay, the visual verifier checks crop replay and string-level answer-claim coupling, and the mathematics verifier checks extracted numeric equalities. These fixed rules verify their stated properties rather than all reasoning steps.

Plug-and-play integration. We expose local verification through adapters and a common credit interface. An adapter extracts the required artifacts and evidence and resolves their step indices. To convert local verification scores into bounded process credits, let B_ℓ be a group of checks assigned to step s_ℓ, with disjoint groups and domain-specific nonnegative weights:

$$q_\ell = \sum_{j \in B_\ell} \omega_{d,j}(x, \tau) v_j, \qquad \omega_{d,j} \geq 0, \qquad \sum_{j=1}^{m} \omega_{d,j} \leq 1. \tag{5}$$

The weights encode normalization and credit eligibility, excluded checks have zero weight. For example, mathematical checks are averaged within internal segments and across scored interaction steps, yielding one credit per scored interaction step. Individually credited artifacts use singleton groups. Unlike the local verification score, its normalization can depend on other artifacts in the trajectory. The complete verifier module for extraction, verification, and credit mapping returns

$$\Phi_d(x,\tau) = \big[(s_\ell, q_\ell)\big]_{\ell=1}^{n}, \qquad s_\ell \in \{1,\dots,T\}, \quad q_\ell \geq 0, \quad \sum_{\ell=1}^{n} q_\ell \leq 1. \tag{6}$$

Here, n is the number of emitted records, zero credits may be omitted, while full audits retain failed checks. Downstream reward composition consumes only process credits and their assigned interaction steps. Compatible verifier module replacement leaves this computation unchanged, and switching the optimizer does not alter the verification rules. Reuse with a different agent requires compatible artifacts and an adapter for its records. Our empirical evaluation concerns multiple domains, model sizes, and GRPO or PPO.

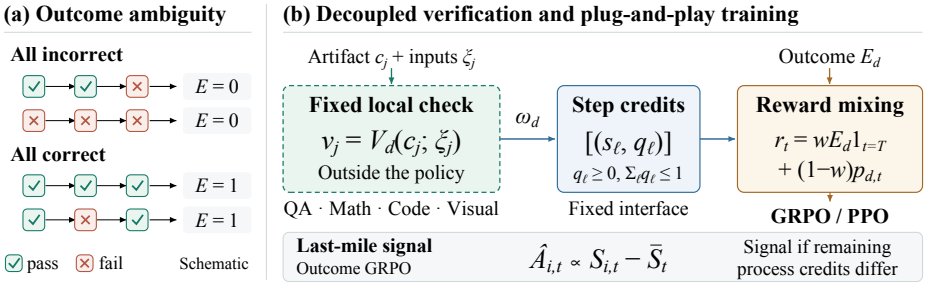

Figure 2: Overview of plug-and-play process verification. (a) Identical outcome rewards can mask differences in intermediate verification results. (b) Local checks are decoupled from policy reasoning and provide step-attributed credits through a common interface. These credits are combined with outcome rewards for GRPO or PPO training. Differences in process returns can provide a last-mile signal within outcome-tied GRPO groups.

3.3 Decoupled Reward Composition

The verifier module connects to training through step-attributed process credits. Let the step-level process reward be $p_{d,t}(x,\tau) = \sum_{\ell:s_\ell=t} q_\ell$ and the trajectory-level process reward be $P_d(x,\tau) = \sum_t p_{d,t}(x,\tau)$, with empty sums equal to zero. For trajectories satisfying terminal eligibility conditions, we assign

$$r_t = w\,\mathbf{1}\{t=T\}E_d(x,\tau) + (1-w)p_{d,t}(x,\tau). \tag{7}$$

The outcome reward is assigned to the terminal interaction step, while process credits retain their assigned interaction steps, including when computed retrospectively. The mixed step rewards sum to Eq. (3), ineligible trajectories receive zero composed reward. GRPO uses undiscounted returns from these rewards, centered and scaled across rollouts sharing the same prompt and step index. Each advantage is applied to that step's generated tokens through the clipped policy objective, with prompt and observation tokens masked. PPO can consume the same rewards.

Setting $w = 1$ recovers outcome-only training under the same agent interface, step normalization, and remaining optimization settings. This is the training-side removal operation: the verifier module's reward contribution is disabled without changing the policy objective. An empty verifier module output instead gives $R_{d,w} = wE_d$; it does not automatically set $w = 1$. Training initially sets $w = 1$ for 50 updates, then activates process rewards by sampling one $w \sim \mathcal{U}(0,1)$ per prompt group and update, shared across its trajectories. Validation uses outcome rewards. Appendix B gives normalization conventions, singleton handling, and objective details.

3.4 LAST-MILE REFINEMENT

Local verification provides additional supervision in groups where outcome-only advantages vanish. We characterize this mechanism before defining how to evaluate the remaining gap between task success and verified intermediate work.

Process signal under tied outcomes. Consider a fixed prompt and its normalization group $\mathcal{G}_t$ at step t, with at least two rollouts. Suppose all members satisfy terminal eligibility, have identical outcome rewards, and share $0 \le w < 1$. With undiscounted returns containing only the rewards in Eq. (7), let $S_{i,t} = \sum_{u=t}^{T_i} p_{d,u}(x, \tau_i)$ denote the remaining process return and $\mathbf{S}_t = (S_{i,t})_{i \in \mathcal{G}_t}$. The normalized reward-derived advantage is

$$\widehat{A}_{i,t} = \frac{S_{i,t} - \mathrm{mean}(\mathbf{S}_t)}{\mathrm{std}(\mathbf{S}_t) + \epsilon_{\mathrm{adv}}/(1-w)}, \tag{8}$$

where $\epsilon_{\mathrm{adv}} > 0$ is the numerical stabilizer. The shared outcome term cancels under centering; scaling the process returns by $1-w$ scales their standard deviation by the same factor. The outcome-only advantages are zero, whereas the mixed advantages remain informative whenever remaining process returns differ. Their dependence on w is negligible when $(1-w)\,\mathrm{std}(\mathbf{S}_t) \gg \epsilon_{\mathrm{adv}}$.

In all-incorrect groups, this signal distinguishes trajectories with different amounts of verified intermediate work. In all-correct groups, it distinguishes successful trajectories whose checked artifacts differ in validity. The latter is the *last-mile* refinement targeted here: improving adherence to intermediate verification criteria even when terminal correctness no longer distinguishes the sampled solutions. The mechanism requires variation in remaining process returns; it neither certifies unchecked reasoning nor guarantees a nonzero aggregate gradient.

Self-consistency and self-reflection. We use *verification-based self-consistency* to denote satisfaction of the applicable verification rules within one trajectory. Let $\mathcal{J}_d(x, \tau)$ index the full set of applicable checks under a fixed evaluation protocol, including failures omitted from credit output; missing required evidence counts as a failed check. Define

$$\begin{aligned} C_d(x, \tau) &= \mathbf{1}\{\mathcal{J}_d(x, \tau) \neq \emptyset \text{ and } v_j = 1 \text{ for all } j \in \mathcal{J}_d(x, \tau)\}\,, \\ M_d(\theta) &= \mathbb{E}_{x \sim \mathcal{D}_d,\, \tau \sim \pi_\theta(\cdot|x)}[E_d(x, \tau)\, C_d(x, \tau)]\,. \end{aligned} \tag{9}$$

For binary E_d, M_d measures joint task success and passage of all applicable checks; for graded E_d, it is a consistency-weighted outcome score. These evaluation quantities are distinct from the graded process reward optimized during training. Comparisons require the same artifact protocol and check coverage across policies.

The evidence requirements also provide an incentive for *self-reflection* through checking and revision. For example, binding successful test evidence to the submitted code version encourages retesting after a revision. This is a behavioral hypothesis, assessed from feedback-dependent revisions in agent trajectories. Retrospective verification alone does not establish such behavior, and an early failed check followed by successful correction still fails the strict C_d criterion. Thus, consistency and reflection require distinct evidence when assessing last-mile refinement.

4 RESULTS

4.1 EXPERIMENTAL SETUP

Training Data and Evaluation Benchmarks. We evaluate our approach across four domains: multi-hop question answering, code generation, multimodal reasoning, and mathematical reasoning. For multi-hop question answering, we train on the training split of **HotpotQA** (Yang et al., 2018) and evaluate on its validation split. For code generation, we train on **TACO** (Li et al., 2023) and evaluate on **LiveCodeBench** (Jain et al., 2025). For multimodal reasoning, we use the **Vision-R1** (Huang et al., 2026) for training and **MATH-Vision** Wang et al. (2024a) for evaluation. For mathematical reasoning, we train on **DeepScaleR-Preview-Dataset** (Luo et al., 2025) and evaluate on **AIME 2025** (Balunović et al., 2025). Training is performed separately for each domain. Within each domain, all compared methods use the same training data and evaluation split.

Methods	AIME 2025	HotpotQA	LiveCodeBench	MATH-Vision	Avg.
Qwen3.5-4B					
ReAct	26.67	46.78	40.47	75.72	47.41
GRPO	33.33	57.73	47.96	79.80	54.71
GRPO + LLM-as-a-judge	36.67	58.28	48.72	**82.01**	56.42
Ours (verification)	**40.00**	**60.18**	**49.57**	81.51	**57.82**
Qwen3.5-9B					
ReAct	36.67	52.56	49.00	79.47	54.43
GRPO	43.33	63.20	56.59	83.19	61.58
Ours (verification)	**46.67**	**66.67**	**58.48**	**84.80**	**64.16**

Table 1: Performance (%) across four domains using Qwen3.5-4B and Qwen3.5-9B. Each domain is trained separately on its corresponding training dataset. Bold numbers indicate the best.

Baselines. We compare our approach with **ReAct**, which uses the initial model without reinforcement learning, and **GRPO**, which trains the same initial model using only the binary outcome reward. We additionally include **GRPO + LLM-as-a-judge**, which uses a frozen **Qwen3.5-9B** judge to provide training rewards for the **Qwen3.5-4B** policy. Across the four domains, the judge evaluates intermediate mathematical reasoning, retrieved evidence, code development and testing, and visual inspection using domain-specific criteria. The resulting process rewards are assigned to the corresponding agent steps and combined with the same binary outcome reward used by GRPO. Our method obtains process rewards from domain-specific verifiers instead of an LLM judge.

Base Models and Training Framework. We use **Qwen3.5-4B** and **Qwen3.5-9B** (Team, 2026) to examine the effectiveness of our approach at two model scales. Our implementation builds on **Agent-R1** (Cheng et al., 2025), with **GRPO** (Shao et al., 2024) and **PPO** (Schulman et al., 2017) as the training algorithms. All methods are implemented using the *verl* package for reward computation, training, and generation. Within each domain and model scale, the RL-trained methods use the same initialization, task interface, tools, fine-tuning method, and training budget. For all domains, we use the first 10,000 examples from the training split for RL training. Qwen3.5-9B uses LoRA with rank 64 and scaling factor $\alpha = 64$. Domain-specific settings are provided in Appendix C.

Evaluation Protocol. We evaluate agents by their final task performance and report accuracy (Acc., %) across all four benchmarks. Accuracy is defined as the percentage of evaluation instances for which the agent produces a correct final answer within the prescribed inference budget. Correctness is assessed using benchmark-specific criteria: answer correctness for AIME 2025 and MATH-Vision, answer exact match for HotpotQA, and passing all evaluation test cases for LiveCodeBench. For interactive tasks, each evaluation episode includes the complete sequence of agent actions and environment feedback until termination, and correctness is determined from the final submitted answer. We use vLLM (Kwon et al., 2023) for speed-up generation.

4.2 Effectiveness Across Domains and Model Scales

We began our experimental analysis with a small yet effective **Qwen3.5-4B** model. We provide a HotpotQA example in Figure 3. As shown in Table 1, our method achieved outcome accuracies of 40.00, 60.18, 49.57, and 81.51 on AIME 2025, HotpotQA, LiveCodeBench, and MATH-Vision, respectively. Compared with outcome-only GRPO, these results represented absolute improvements of 6.67, 2.45, 1.61, and 1.71 percentage points. Across the four benchmarks, our method achieved an average accuracy of 57.82, compared with 54.71 for GRPO and 47.41 for ReAct. We further compared against a process-reward baseline that used Qwen3.5-9B as an LLM judge during GRPO training of the Qwen3.5-4B agent. Our method also achieved a higher

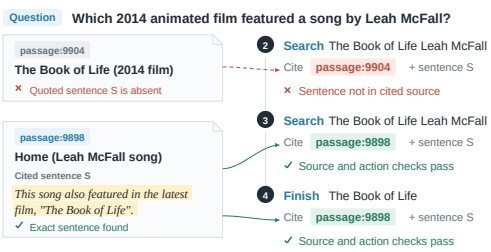

Figure 3: An example of our plug-and-play verifier on HotpotQA.

average accuracy than this baseline (57.82 vs. 56.42), with improvements on three of the four benchmarks. Together, these results provided initial evidence that plug-and-play verification could supply useful process supervision beyond outcome rewards.

Next, we extended our evaluation to the **Qwen3.5-9B** model to examine whether these benefits persisted at a larger policy model scale. Our method outperformed outcome-only GRPO across all four benchmarks, achieving an average accuracy of 64.16, compared with 61.58 for GRPO. On AIME 2025 and HotpotQA, our method achieved 46.67 and 66.67, exceeding GRPO by 3.34 and 3.47 percentage points. The improvements also extended to code generation and multimodal reasoning, with gains of 1.89 percentage points on LiveCodeBench and 1.61 on MATH-Vision. The consistent superior performance at both model scales suggested that the benefit of process verification was not confined to a single domain or policy model size.

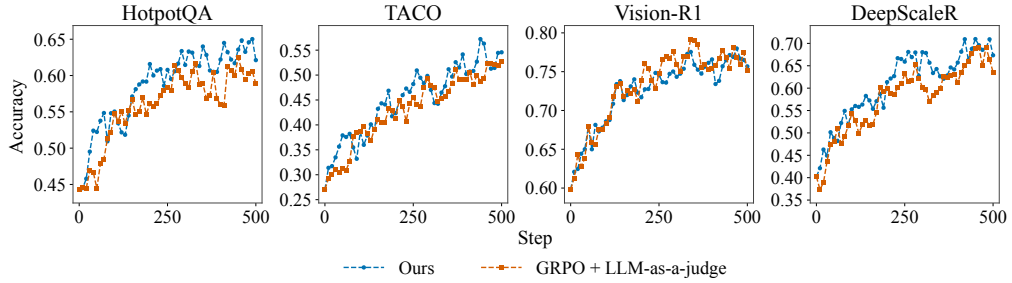

Figure 4: Training accuracy of Qwen3.5-4B with Ours and GRPO with LLM-as-a-judge.

Regarding the comparison with LLM-based process rewards, our method also outperforms the LLM-as-a-judge baseline on average. With Qwen3.5-4B, it achieved higher accuracies on AIME 2025 (40.00 vs. 36.67), HotpotQA (60.18 vs. 58.28), and LiveCodeBench (49.57 vs. 48.72), and slightly lower on MATH-Vision (81.51 vs. 82.01). These results supported the effectiveness of our verification process supervision, which improved outcome accuracy over GRPO across all four domains and achieved a higher average accuracy than the LLM-as-a-judge baseline without relying on a strong and powerful LLM judge to compute process rewards. Overall, these findings supported the applicability of our modular framework, which decoupled process verification from reward composition and policy optimization to integrate domain-specific verifiers through a shared interface.

4.3 Ablation Studies

Having established the above results, we examined whether verification-based process feedback improved agent performance and which verification mechanisms contributed to the improvement. We focused on **HotpotQA**, which provided both reference answers and annotated supporting facts, allowing outcome supervision and verification-based process supervision to be examined separately. All results in this subsection used **Qwen3.5-4B** with **GRPO**. We reported accuracy based on normalized exact match between the agent's final answer and the reference answer.

(a) Verification mechanisms

Configuration	Acc. (%)
GRPO (outcome only)	57.73
+ LLM-as-a-judge	58.28
+ plug-and-play verification (ours)	60.18
+ Gold evidence (upper bound)	61.40

(b) Reward configurations

Configuration	Acc. (%)
ReAct	46.78
Outcome only (GRPO)	57.73
Process only	47.27
Outcome + process	61.40

Table 2: Ablation studies on HotpotQA with Qwen3.5-4B. (a) Each verification mechanism adds process rewards to the outcome-only GRPO baseline. (b) We compare outcome rewards, process rewards, and their combination under GRPO, with ReAct as a reference. Process rewards in (b) use verification against gold evidence.

Effect of process-verification feedback. To examine the interaction between outcome and process supervision, we first considered outcome-only GRPO, which improved performance from the Re-

Act baseline of 46.78 to 57.73. We then used process rewards based on gold-evidence matching to isolate the effect of process supervision. As shown in Table 2, process-only training achieved 47.27, suggesting that process supervision alone was insufficient to optimize the final task objective. In contrast, combining process and outcome rewards further improved performance to 61.40. Since outcome rewards directly supervise final-answer correctness, whereas process rewards provide feedback on the intermediate evidence produced along the trajectory, these results suggest that evidence-based process supervision is most effective as a complementary signal rather than a replacement for the final task objective. We next compared different sources of process supervision, including our verifier that does not rely on gold evidence.

Comparison of Verification Mechanisms. We compare three process verification mechanisms. As shown in Table 2, our verifier achieves an accuracy of 60.18, outperforming LLM-as-a-judge. Gold-evidence matching achieves the accuracy at 61.40. These results suggest that deterministic certificate verification can provide useful process supervision without gold evidence or an LLM judge, achieving accuracy within 1.22 percentage points of gold-evidence matching.

4.4 Effectiveness Across Policy Optimization Algorithms

Finally, we evaluated our framework with both GRPO and PPO on HotpotQA using Qwen3.5-4B to examine whether the benefits of process verification extended beyond GRPO. For each algorithm, we compared outcome-only training with training augmented by verification process rewards. We used the same verifier and reward-mixing schedule across the two algorithms, while keeping initialization, task interface, training budget, and evaluation settings fixed.

As shown in Table 3, incorporating process verification improved outcome accuracy from 57.73 to 60.18 under GRPO and from 55.10 to 57.60 under PPO, yielding absolute gains of 2.45 and 2.50 percentage points, respectively. Despite their different baseline accuracies, both algorithms benefited from the same verification module. These results supported the plug-and-play design of our framework: decoupling process verification from policy optimization allowed the verifier to be reused across GRPO and PPO while retaining its benefit on HotpotQA.

Algorithm	Outcome-only	+ Verification
GRPO	57.73 (+2.45)	**60.18**
PPO	55.10 (+2.50)	**57.60**

Table 3: Effect of process verification under GRPO and PPO on HotpotQA with Qwen3.5-4B. We report outcome accuracy (%). "+ Verification" adds verification-based process rewards to outcome-only training. Values in parentheses denote absolute accuracy gains over outcome-only training within each algorithm, in percentage points.

5 Conclusion

We propose a plug-and-play verifier framework that enables process-supervised RL for language model agents by decoupling process verification from reward composition and policy optimization. We note that our framework shares a similar idea with recent works. Yuan et al. (2026) convert symbolic and algorithmic oracles into turn-level process rewards. Xu et al. (2026) attribute verified evidence to the search actions that first exposed it. However, our framework is unique in combining a unified contract of bounded, step-attributed credits, deterministic verifiers that require neither gold evidence nor LLM judges, and group-shared reward mixing. Our results illustrate the potential of explicit process verification as a modular complement to outcome rewards in agentic RL. **Limitations and future work.** We evaluate our proposed verifiers by mostly checking per-turn reasoning step, such as evidence grounding and execution replay, rather than the multiple-turn semantic correctness of intermediate steps. We will incorporate process reward models (Lightman et al., 2024; Setlur et al., 2025) and the most recent calibrated decision models such as Jev for future expanding of plug-and-play verifier. Second, our agents operate within short interaction horizons on public agentic benchmarks. We will extend our framework to much longer-horizon environments (Zhou et al., 2024; Jimenez et al., 2024) for production agentic post-training.

DISCLOSE USE OF LLMS

We used LLMs, such as Claude and GPT, for initial idea brainstorming and later to check grammar and spelling and improve the clarity of our manuscript's language. All scientific contents, including figures, tables, experimental design, analysis, and interpretations, are entirely the original work.

ETHICS STATEMENT

All authors of this study strictly adhere to the ICLR ethical guidelines. Our research does not involve any potential conflicts of interest or sponsorship issues. We have carefully considered and addressed issues related to discrimination, bias, and fairness in our methodology. This study does not involve any privacy or security concerns, fully complies with legal requirements, and upholds the highest standards of research integrity. All experimental procedures and data processing follow established ethical guidelines for machine learning research.

REPRODUCIBILITY STATEMENT

To support reproducibility of the results, we describe the proposed plug-and-play verification framework and decoupled reward composition. The experimental settings and evaluation protocol are presented in Section 4.1, while dataset specifications, training hyperparameters, and policy optimization details are documented in Appendices A and C . The complete code will be released upon acceptance of the paper at `https://github.com/momo1443/plug-and-play-verification`.

REFERENCES

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Jiajun Chai, Guojun Yin, Zekun Xu, Chuhuai Yue, Yi Jia, Siyu Xia, Xiaohan Wang, Jiwen Jiang, Xiaoguang Li, Chengqi Dong, et al. Rlfactory: A plug-and-play reinforcement learning post-training framework for llm multi-turn tool-use. *arXiv preprint arXiv:2509.06980*, 2025.

Mingyue Cheng, Shuo Yu, Daoyu Wang, Qingchuan Li, Xiaoyu Tao, Jie Ouyang, Yucong Luo, Yitong Zhou, Qi Liu, and Enhong Chen. Agent-r1: A unified and modular framework for agentic reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

Yu Cui, Hang Fu, Haibin Zhang, Licheng Wang, and Cong Zuo. Free-mad: Consensus-free multi-agent debate. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 31977–31997, 2026.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-rlvr: Training software engineering agents via guidance and environment rewards. *arXiv preprint arXiv:2506.11425*, 2025.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026a.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026b.

Shubham Gandhi, Saurabh Goyal, Kiran Kate, and Yara Rizk. Draco: Fine-grained credit assignment with dynamic rubrics for long-horizon agent training. *arXiv preprint arXiv:2609.04094*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

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, Xin Cheng, Lei Feng, Bo An, et al. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. In *International Conference on Learning Representations*, volume 2026, pp. 27572–27593, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Wenxuan Huang, Bohan Jia, Shaosheng Cao, Zheyu Ye, Zhe Xu, Yao Hu, Shaohui Lin, et al. Vision-r1: Incentivizing reasoning capability in multimodal large language models. In *International Conference on Learning Representations*, volume 2026, pp. 63794–63812, 2026.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

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 *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Yijun Lu, Rui Ye, Jiajun Wang, Yuwen Du, Tian Jin, Songhua Liu, and Siheng Chen. Abseeker: Training long-horizon search agents via answer-backtracked credit assignment. *arXiv preprint arXiv:2608.05102*, 2026.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 3(5), 2025.

Massimiliano Pronesti, Anja Belz, and Yufang Hou. Beyond outcome verification: Verifiable process reward models for structured reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32187–32202, 2026.

Jiacheng Sang, Mengyuan Li, Sanxing Chen, Yukun Huang, Yu Feng, and Bhuwan Dhingra. Searchatlas: Analyzing agentic search strategies via evidential query graphs. *arXiv preprint arXiv:2609.10901*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 60808–60838, 2025.

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.

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.

Qwen Team. Qwen3. 5: Towards native multimodal agents. *URL: https://qwen. ai/blog*, 2026.

Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. Information gain-based policy optimization: A simple and effective approach for multi-turn llm agents. *arXiv e-prints*, pp. arXiv–2510, 2025a.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *Advances in Neural Information Processing Systems*, 37:95095–95169, 2024a.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024b.

Xiaoxuan Wang, Han Zhang, Haixin Wang, Yidan Shi, Ruoyan Li, Kaiqiao Han, Chenyi Tong, Haoran Deng, Renliang Sun, Alexander Taylor, et al. Arlarena: A unified framework for stable agentic reinforcement learning. *arXiv preprint arXiv:2602.21534*, 2026a.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, et al. Ragen-2: Reasoning collapse in agentic rl. *arXiv preprint arXiv:2604.06268*, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Ke Xu, Han Xu, Xinran Chen, Yuqian Wang, Zhixuan Li, Xiaojian Liu, Changwo Wu, Jianqiang Xia, and Yuchen Li. Stamp: Provenance-guided credit assignment for deep search agents. *arXiv preprint arXiv:2607.11172*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Kaibing Yang, Guangfeng Cai, Shengtian Yang, Shuo He, Yu Li, Mengyi Liu, Pengwei Chen, Jun Xu, and Lei Feng. Progress-conditioned group policy optimization for long-horizon agentic tasks. *arXiv preprint arXiv:2607.22724*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

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*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Huining Yuan, Zelai Xu, Huaijie Wang, Xiangmin Yi, Jiaxuan Gao, Xiao-Ping Zhang, Yu Wang, Chao Yu, and Yi Wu. Verifiable process rewards for agentic reasoning. *arXiv preprint arXiv:2605.10325*, 2026.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Rongzhi Zhu, Xiangyu Liu, Yi Liu, Shuo Zhang, Ruirui Zhang, Rui Wu, Tao Jiang, Zequn Sun, Wenhao Xu, and Wei Hu. Dense process supervision for search agents via fact utility estimation. *arXiv preprint arXiv:2609.00833*, 2026.

A Training and Evaluation Benchmarks

We conduct training and evaluation using benchmarks from four domains: multi-hop question answering, code generation, multimodal reasoning, and mathematical reasoning.

- HotpotQA Yang et al. (2018): a large-scale multi-hop question answering dataset constructed from Wikipedia. The dataset contains 90,447 training and 7,405 validation examples, with questions requiring reasoning over information distributed across multiple documents. Each example is additionally annotated with sentence-level supporting facts, enabling explicit evaluation of evidence identification and multi-hop reasoning.
- TACO Li et al. (2023): a corpus consisting of 25,443 competitive programming problems from the training split of the TACO dataset. Each problem is paired with a natural-language problem statement, input-output specifications, reference solutions, and executable test cases, spanning algorithmic topics and difficulty levels for code generation and reasoning.
- LiveCodeBench Jain et al. (2025): a 1,055-problem code generation benchmark consisting of competitive programming problems collected from LeetCode, AtCoder, and Codeforces. We use the `code_generation_lite` setting with `release_v6`, covering problems released between May 2023 and April 2025. Its temporally recent problems and hidden test cases make it suitable for evaluating coding ability with reduced contamination risk. We use LiveCodeBench for evaluation to measure the generalization of code generation.
- Vision-R1-RL Huang et al. (2026): a multimodal mathematical reasoning dataset containing approximately 11.3K image-text problems designed for reinforcement learning with verifiable rewards. The dataset consists primarily of visually grounded mathematical and geometric reasoning problems with deterministic final answers.
- MATH-Vision Wang et al. (2024a): a multimodal mathematical reasoning benchmark consisting of 3,040 visual mathematics problems collected from real-world mathematical competitions. The benchmark spans 16 mathematical disciplines and 5 difficulty levels, covering diverse problems that require both visual understanding and reasoning. We use MATH-Vision for evaluation to measure the generalization of multimodal reasoning.
- DeepScaleR-Preview-Dataset Luo et al. (2025): a curated training corpus of approximately 40,000 unique mathematics problem–answer pairs compiled from AIME problems (1984–2023), AMC problems (before 2023), the Omni-MATH dataset, and the Still dataset, spanning difficulty levels from high-school competition to advanced mathematical reasoning.
- AIME 2025 Balunović et al. (2025): a 30-problem benchmark drawn from the 2025 AIME I and AIME II exams, it is commonly used as a fresher benchmark with lower contamination risk for models whose a recent competition benchmark.

The downloading link for each dataset can be accessed from the Hugging Face website:

- HotpotQA: `https://huggingface.co/datasets/hotpotqa/hotpot_qa`
- TACO: `https://huggingface.co/datasets/BAAI/TACO`
- LiveCodeBench: `https://huggingface.co/datasets/livecodebench/code_generation_lite`
- Vision-R1-RL: `https://huggingface.co/datasets/Osilly/Vision-R1-rl`
- MATH-Vision: `https://huggingface.co/datasets/MathLLMs/MathVision`
- DeepScaleR-Preview-Dataset: `https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset`
- AIME 2025: `https://huggingface.co/datasets/MathArena/aime_2025`

B Policy Optimization Details

We describe terminal verification, step-level reward construction, and advantage computation. The same policy objective and token masking are used across verifier modules.

Terminal verification and credit backfill. In the HotpotQA implementation, the selected verifier is invoked after a trajectory terminates and evaluates the recorded trajectory. The module returns a `VerificationResult` containing bounded credits and their source step indices. These credits are assigned back to the corresponding recorded policy transitions, yielding $p_{d,t}(x, \tau_i)$ as defined in Section 3.3. Verification time and reward attribution are therefore distinct: a credit computed after termination can be assigned to an earlier interaction step. The outcome reward is assigned to the final policy transition. The resulting step rewards are passed to advantage computation.

Step-level reward construction. Fix an optimizer update k and a prompt group g with input x. We sample G trajectories $\{\tau_i\}_{i=1}^{G}$ using the rollout policy π_{old}. Let T_i be the number of policy steps in τ_i, $E_i = E_d(x, \tau_i)$, and $p_{i,t} = p_{d,t}(x, \tau_i)$. The step rewards implement the schedule in Eq. equation 10:

$$r_{i,t} = \begin{cases} \mathbf{1}\{t = T_i\} E_i, & k \le K_{\text{w}}, \\ w_{k,g} \mathbf{1}\{t = T_i\} E_i + (1 - w_{k,g}) p_{i,t}, & k > K_{\text{w}}. \end{cases} \tag{10}$$

Here $\mathbf{1}\{\cdot\}$ is the indicator function. The weight $w_{k,g}$ is shared by all trajectories in group g; we suppress k and the fixed group g in $r_{i,t}$ for readability. Since $\sum_t p_{i,t} = P_d(x, \tau_i)$, the step rewards sum to the trajectory-level training reward $R_{k,i}$.

Reproducible group-shared mixing. After warmup, the implementation deterministically generates the pseudorandom uniform mixing weight from a hash of `(global_step, prompt_group_key)`. All trajectories sharing this pair receive the same weight, making reward mixing reproducible for a given update and prompt group. The run manifest records this schedule as `uniform_0_1_per_prompt_group`. During the first 50 updates and validation, $w = 1$; in HotpotQA, the reward is terminal EM only, without process or format bonuses.

Group-relative advantages. We use discount factor $\gamma = 1$ and compute the undiscounted reward-to-go:

$$R_{i,t} = \sum_{u=t}^{T_i} r_{i,u}. \tag{11}$$

Each reward contributes only to the return of its own step and all preceding steps, regardless of when verification is performed.

For this fixed prompt, let $\mathcal{G}_t = \{i : T_i \ge t\}$ and $n_t = |\mathcal{G}_t|$. Normalization is performed separately for each prompt and step index. Trajectories that terminate before step t are excluded from that step's statistics. For each nonempty step group, we use

$$\mu_t = \begin{cases} \dfrac{1}{n_t} \displaystyle\sum_{i \in \mathcal{G}_t} R_{i,t}, & n_t > 1, \\ 0, & n_t = 1, \end{cases}$$
$$\sigma_t = \begin{cases} \sqrt{\dfrac{1}{n_t - 1} \displaystyle\sum_{i \in \mathcal{G}_t} (R_{i,t} - \mu_t)^2}, & n_t > 1, \\ 1, & n_t = 1. \end{cases} \tag{12}$$

We use the sample standard deviation with Bessel correction for groups containing multiple trajectories; singleton groups use the uncentered return with unit scale. The step-level advantage is

$$\hat{A}_{i,t} = \frac{R_{i,t} - \mu_t}{\sigma_t + \epsilon_{\text{adv}}}, \tag{13}$$

where $\epsilon_{\text{adv}} > 0$ stabilizes the denominator. For a singleton step group, this convention gives $\hat{A}_{i,t} = R_{i,t}/(1 + \epsilon_{\text{adv}})$, retaining the return instead of centering it to zero. Outcome and process rewards are mixed before computing returns and applying this single normalization; they are not normalized separately. Advantages are held fixed during policy optimization.

Policy tokens and environment observations. Let $\mathcal{T}_i$ denote the set of valid policy steps in τ_i. For each $t \in \mathcal{T}_i$, let $\mathcal{I}_{i,t}$ contain the positions of valid policy-generated tokens in step t, with $N_{i,t} = |\mathcal{I}_{i,t}| > 0$. These tokens include generated reasoning, tool-call arguments, and the final answer. Prompt tokens and environment observations are excluded from both the policy surrogate and the KL regularizer. Environment observations remain in the conditioning context for subsequent policy generations.

For a policy-generated token $z_{i,n}$, let $h_{i,n}$ denote its conditioning context. All tokens belonging to step t receive the same advantage:

$$\widetilde{A}_{i,n} = \hat{A}_{i,t}, \qquad n \in \mathcal{I}_{i,t}. \tag{14}$$

The token-level importance ratio is

$$\rho_{i,n}(\theta) = \frac{\pi_\theta(z_{i,n} \mid h_{i,n})}{\pi_{\text{old}}(z_{i,n} \mid h_{i,n})}. \tag{15}$$

Clipped policy objective. We first define the clipped surrogate

$$\ell_{i,n}^{\text{clip}}(\theta) = \min\Big\{\rho_{i,n}(\theta)\widetilde{A}_{i,n}, \text{clip}(\rho_{i,n}(\theta), 1-\epsilon, 1+\epsilon)\,\widetilde{A}_{i,n}\Big\}, \tag{16}$$

where $\epsilon > 0$ is the clipping threshold. Dual clipping additionally bounds the surrogate for negative advantages:

$$\ell_{i,n}(\theta) = \begin{cases} \max\Big\{\ell_{i,n}^{\text{clip}}(\theta), c\widetilde{A}_{i,n}\Big\}, & \widetilde{A}_{i,n} < 0, \\ \ell_{i,n}^{\text{clip}}(\theta), & \widetilde{A}_{i,n} \geq 0, \end{cases} \tag{17}$$

with $c = 3$.

For each sampled group, we maximize

$$\widehat{\mathcal{J}}(\theta) = \frac{1}{\sum_{i=1}^{G}|\mathcal{T}_i|}\sum_{i=1}^{G}\sum_{t\in\mathcal{T}_i}\frac{1}{N_{i,t}}\sum_{n\in\mathcal{I}_{i,t}}\ell_{i,n}(\theta) - \beta\widehat{\mathcal{L}}_{\text{KL}}(\theta), \tag{18}$$

where $\beta \geq 0$ controls KL regularization. The surrogate is averaged first over valid policy tokens within each interaction step and then over valid interaction steps in the sampled group.

For KL regularization, we define

$$\begin{aligned} d_{i,n}(\theta) &= \log \pi_{\text{ref}}(z_{i,n} \mid h_{i,n}) - \log \pi_\theta(z_{i,n} \mid h_{i,n}), \\ \mathcal{K}_{i,n}(\theta) &= \exp\big(d_{i,n}(\theta)\big) - d_{i,n}(\theta) - 1. \end{aligned} \tag{19}$$

This penalty is evaluated on rollout tokens and uses the same averaging scheme as the policy surrogate:

$$\widehat{\mathcal{L}}_{\text{KL}}(\theta) = \frac{1}{\sum_{i=1}^{G}|\mathcal{T}_i|}\sum_{i=1}^{G}\sum_{t\in\mathcal{T}_i}\frac{1}{N_{i,t}}\sum_{n\in\mathcal{I}_{i,t}}\mathcal{K}_{i,n}(\theta). \tag{20}$$

The reference policy π_{ref} remains fixed throughout training, whereas π_{old} is the policy used to collect the current rollouts. KL regularization is applied only in the optimization objective and does not enter the task rewards, reward-to-go, or advantage normalization.

C Training Details

Infrastructure. All experiments are conducted on a single Kubernetes (K8s) node equipped with 8× NVIDIA A40 48GB GPUs. We use PyTorch with Fully Sharded Data Parallelism (FSDP) for distributed training and vLLM (server mode) for rollout generation. It takes around 37 hours to run for 500 steps training on 4B models and 80 hours on 9B models, respectively. It takes 15 minutes for vLLM inferences.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	20
GRPO group size	4 rollouts
Mini-batch size	20 prompts
Optimization epochs per rollout	1
Max prompt length	2048 tokens
Max completion length	4096 tokens
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
Random seed	42

Table 4: Hyperparameters for Qwen3.5-4B GRPO training on DeepScaleR.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	20
GRPO group size	4 rollouts
Mini-batch size	20 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	1024 tokens
Maximum agent turns	4
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
Random seed	42

Table 5: Hyperparameters for Qwen3.5-4B GRPO training on HotpotQA.

Hyperparameter	Value
Fine-tuning method	LoRA ($r = 64$, $\alpha = 64$)
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	20
GRPO group size	4 rollouts
Mini-batch size	20 prompts
Optimization epochs per rollout	1
Max prompt length	2048 tokens
Max completion length	5120 tokens
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
LoRA target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `gate_proj`, `down_proj`
Random seed	42

Table 6: Hyperparameters for Qwen3.5-9B LoRA GRPO training on DeepScaleR.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	20
GRPO group size	4 rollouts
Mini-batch size	20 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	2048 tokens
Maximum agent turns	5
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
Random seed	42

Table 7: Hyperparameters for Qwen3.5-4B GRPO training on TACO.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	8
GRPO group size	4 rollouts
Mini-batch size	8 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	2048 tokens
Maximum agent turns	3
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
Random seed	42

Table 8: Hyperparameters for Qwen3.5-4B GRPO training on Vision-R1.

Hyperparameter	Value
Fine-tuning method	LoRA ($r = 64$, $\alpha = 64$)
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	8
GRPO group size	4 rollouts
Mini-batch size	8 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	1024 tokens
Maximum agent turns	4
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
LoRA target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `gate_proj`, `down_proj`
Random seed	42

Table 9: Hyperparameters for Qwen3.5-9B LoRA GRPO training on HotpotQA.

Hyperparameter	Value
Fine-tuning method	LoRA ($r = 64$, $\alpha = 64$)
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	20
GRPO group size	4 rollouts
Mini-batch size	20 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	2048 tokens
Maximum agent turns	5
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
LoRA target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `gate_proj`, `down_proj`
Random seed	42

Table 10: Hyperparameters for Qwen3.5-9B LoRA GRPO training on TACO.

Hyperparameter	Value
Fine-tuning method	LoRA ($r = 64$, $\alpha = 64$)
Optimizer	AdamW
Learning rate	1e-6
Training steps	500
LR schedule	Constant; no warmup
Global prompt batch size	8
GRPO group size	4 rollouts
Mini-batch size	8 prompts
Optimization epochs per rollout	1
Max prompt length	8192 tokens
Max completion length	2048 tokens
Maximum agent turns	3
Sampling temperature	1.0
Top-p / Top-k	1.0 / -1
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.001
Precision	bfloat16
LoRA target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `gate_proj`, `down_proj`
Random seed	42

Table 11: Hyperparameters for Qwen3.5-9B LoRA GRPO training on Vision-R1.

D Example Responses and Verification Traces

We present one example from each of the four evaluation datasets, covering multi-hop question answering, code generation, multimodal reasoning, and mathematical reasoning, in that order. Each example shows the task, a response, and the intermediate artifacts used for verification.

D.1 Multi-hop Question Answering: HotpotQA

Question: What 2014 American 3D computer-animated musical fantasy adventure comedy film features a song recorded by British singer Leah McFall?

Reference answer: The Book of Life.

Recorded trajectory (condensed).
Step 1: Initial search. The agent searched for "2014 American 3D computer-animated musical fantasy adventure comedy film song recorded by British singer Leah McFall". The retrieved results included `passage:9904`, *The Book of Life (2014 film)*, and `passage:9898`, *Home (Leah McFall song)*. The latter contained the sentence

> "This song also featured in the latest film, `"The Book of Life"`."

We denote this sentence by S. No certificate was required for the initial search.

Step 2: Search with a misattributed citation. The agent searched for "The Book of Life Leah McFall". Its certificate cited `passage:9904`, quoted S, and specified `"The Book of Life"` as the target. However, S was absent from the cited passage.

Step 3: Search with a matching source. The query, support span, and target were unchanged, but the certificate now cited `passage:9898`, which contained S.

Step 4: Final submission. The agent submitted "The Book of Life", citing `passage:9898` and S, with `"The Book of Life"` as the answer span. The submitted answer matched the reference.

D.2 Code Generation: LiveCodeBench

Task (paraphrased): Given an integer array, find the maximum sum of a nonempty contiguous segment whose adjacent elements have opposite parity.

Selected public cases: $[1,2,3,4,5] \mapsto 15$; $[9,9,8,8] \mapsto 17$; $[-1000] \mapsto -1000$.

Constructed response.
Maintain the largest sum of a valid segment ending at each position. Extend the preceding segment only if the adjacent elements have opposite parity; otherwise, start a new segment. Initialize with the first element to enforce the nonempty requirement, including when all elements are negative.

```
best = current = a[0]
for i in range(1, len(a)):
    if a[i] % 2 != a[i - 1] % 2:
        current = max(a[i], current + a[i])
    else:
        current = a[i]
    best = max(best, current)
```

Output: `best`. The recurrence takes $O(n)$ time and $O(1)$ auxiliary space.

D.3 Multimodal Reasoning: MATH-Vision

Question (paraphrased): Using the balanced scale below, determine how many plums have the same weight as one apple.
Reference answer: 3.

Constructed response.
The left pan contains three apples and two plums, and the right pan contains one apple and eight plums. Let A and P denote the weights of an apple and a plum, respectively. The balance gives

$$3A + 2P = A + 8P, \qquad 2A = 6P, \qquad A = 3P.$$

Thus, one apple weighs as much as three plums.

Final answer: $\boxed{3}$.

Visual evidence. On the original 765×225 image, we constructed a left-pan crop with bounding box $(35, 25, 304, 131)$ and a right-pan crop with bounding box $(474, 28, 734, 130)$, in $(\text{left}, \text{top}, \text{right}, \text{bottom})$ pixel coordinates. The certificate cited both crop identifiers and included the answer 3 in its claim.

D.4 Mathematical Reasoning: AIME 2025

Question: Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.

Reference answer: 70.

Constructed response.
The base-b values are $17_b = b + 7$ and $97_b = 9b + 7$. Set $d = b + 7$. Since

$$9 \times 7 - 7 = 56,$$

we have $9b + 7 = 9d - 56$, so d must divide 56. The condition $b > 9$ requires $d > 16$, leaving the divisors 28 and 56. The corresponding bases are

$$28 - 7 = 21, \qquad 56 - 7 = 49.$$

Both satisfy the original divisibility condition, and their sum is

$$21 + 49 = 70.$$

Final answer: $\boxed{70}$.

Shared interface. The four examples expose different verification objects: source-grounded certificates, code versions with execution evidence, replayable image crops, and numerical equalities. The domain-specific verifiers map their checks to the shared `VerificationResult` contract of bounded credits and source-step indices. This common interface lets the reward composer consume process feedback without inspecting the underlying artifact type, while the outcome signal separately evaluates the final answer.