

STEP-VERIFIER: Learning to Verify Intermediate Agent Behavior with Process Reward Models

Longhui Zhang

University of Hong Kong

Xian Wu

University of Hong Kong <https://orcid.org/0009-0000-1826-1434>

Chunyu Chen

Xiaomi (China)

Fen Huang

Xiaomi (China)

Bin Wang

wangbin11@xiaomi.com

Xiaomi (China) <https://orcid.org/0009-0004-2911-9715>

Research Article

Keywords: LLM, Agent

Posted Date: October 21st, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11444038/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

STEP-VERIFIER: Learning to Verify Intermediate Agent Behavior with Process Reward Models

Longhui Zhang¹ Xian Wu¹ Chunyuan Chen² Fen Huang² Bin Wang^{2,*}

¹Hongkong University

²Xiaomi Inc.

*Corresponding author: wangbin11@xiaomi.com

Abstract

Large language model agents complete tasks through long sequences of tool calls, yet most training signals only indicate whether the entire attempt succeeded. We explore whether a separate verifier can learn to judge the quality of individual actions before an episode finishes. Our system, STEP-VERIFIER, scores each environment-observation and action pair instead of whole trajectories. It is trained on automatically synthesized comparisons distilled from a success-labeled corpus of logged interactions: Monte-Carlo return differences between states sharing a common prefix yield pairwise supervision, and a Bradley-Terry ranking head converts these pairs into calibrated per-action scores. While the policy is being learned, the verifier filters low-quality interaction logs before each update and provides dense intermediate feedback that is annealed toward zero as training progresses, so the final policy never relies on the verifier at deployment. Using Llama-3.1-8B-Instruct and Mistral-7B-Instruct backbones, STEP-VERIFIER outperforms training from final scores alone across GAIA, ToolBench, and AgentBoard, and its per-action scores align substantially better with human judgments than trajectory-level scoring baselines.

1 Introduction

A tool-using language-model agent succeeds or fails over the course of dozens of interactions: it selects an API, parses the response, reformulates a query, recovers from an error, and only at the end of the episode discovers whether the whole chain accomplished the task. Outcome-rewarded training mirrors this structure from the learning side. Group-relative methods such as GRPO score each sampled attempt with a single terminal signal and derive one normalized advantage per trajectory (Shao et al., 2024), a scheme that is cheap, verifiable, and remarkably effective, and that now underlies much of agentic RL (Guo et al., 2025; Yu et al., 2026; Wang et al., 2025; Chen et al., 2025; Luo et al., 2025). Its blind spot is equally structural: the only judge of intermediate behavior is the final outcome, so an episode that fails because of one wrong call in step two and an episode that fails everywhere are indistinguishable to the learner.

One could ask a model to judge the intermediate behavior directly. Process reward models supply exactly this in mathematical reasoning, scoring individual solution steps rather than final answers (Lightman et al., 2024; Cui et al., 2025), and reward models learned from comparisons are the backbone of preference training (Ouyang et al., 2022; Rafailov et al., 2023). Transplanted to agentic settings, however, both strands inherit their original costs: step-level labels come from human annotation or trained annotators whose quality caps the final verifier, and learned critics for long-horizon interaction are notoriously sample-inefficient. The open question is whether an action-level verifier for agents can be trained with no human step labels and no extra environment interaction, using only material that outcome-rewarded training already throws away.

We find that the discarded material is sufficient. Logged interactions from outcome-rewarded training carry terminal success labels, and the same task is typically attempted many times,

so the corpus contains multiple continuations from semantically identical interaction prefixes. When two states share a common prefix, the difference between the average final return of their logged continuations is a Monte-Carlo estimate of how much better one branch of the interaction was than the other—a pairwise comparison between the two actions that split the prefix, obtained without a single fresh rollout. Aggregated over a success-labeled corpus, these comparisons densely supervise an action-level verifier, and a Bradley-Terry ranking head (Bradley & Terry, 1952) converts them into calibrated scores.

STEP-VERIFIER realizes this design. The verifier consumes each environment-observation and action pair together with a truncated window of recent interaction and produces a scalar score. It is trained purely on synthesized prefix-split comparisons, with two guarantees we prove: the Monte-Carlo return difference is an unbiased and consistent estimator of the underlying value gap, and under a Bradley-Terry response model the population training objective is minimized exactly when the verifier’s score gaps recover those value gaps, so the scores are calibrated rather than merely ordered. During policy training the verifier plays two roles and then exits: before each update it filters the lowest-scoring interaction logs from the batch, and during the advantage computation it injects dense per-action feedback whose mixing weight is annealed linearly to zero, so the final phase of training is indistinguishable from outcome-only optimization and the deployed policy carries no verifier dependency. Our contributions are:

- an action-level process verifier for tool-using agents trained entirely from a success-labeled log corpus, whose pairwise supervision is synthesized from Monte-Carlo return differences between states sharing a common prefix—no human step labels, reward models, or additional rollouts;
- a verifier-in-the-loop training scheme combining pre-update log filtering with dense intermediate feedback annealed to zero, together with proofs that the synthesized labels are unbiased value-gap estimates and that the Bradley-Terry head yields calibrated per-action scores;
- evaluations with Llama-3.1-8B-Instruct and Mistral-7B-Instruct on GAIA, ToolBench, and AgentBoard, where STEP-VERIFIER outperforms outcome-only, trajectory-verifier, and binary-process-verifier baselines on every aggregate metric, and its per-action scores align with human judgments far better than trajectory-level scoring.

2 Related Works

Outcome-Rewarded RL for Tool-Using Agents Training agents against verifiable terminal rewards has become the default recipe for multi-turn tool use, embodied by GRPO and its descendants (Shao et al., 2024; Guo et al., 2025; Yu et al., 2026) and extended to interactive and hierarchical settings (Wang et al., 2025; Chen et al., 2025; Luo et al., 2025; Zhou et al., 2024). These methods differ in how they normalize or decompose the terminal signal—group-relative baselines, turn-level value functions, within-state grouping (Feng et al., 2026)—but the signal itself remains trajectory-scalar. STEP-VERIFIER is orthogonal: it leaves the outcome objective untouched and adds a separately learned action-level judge whose influence is removed before training ends.

Outcome and Process Reward Models Reward models trained from comparisons underpin preference-based post-training (Ouyang et al., 2022; Rafailov et al., 2023), and process reward models adapt the idea to intermediate reasoning steps (Lightman et al., 2024), with implicit process rewards recovering step scores from outcome-labeled data (Cui et al., 2025). Applied to agents, trajectory-level reward models score whole attempts and inherit the granularity problem of the outcome signal itself, while process variants in reasoning depend on step annotations or solve-state heuristics specific to mathematics. STEP-VERIFIER synthesizes its step-level comparisons from the interaction structure of logged agent trajectories—prefix sharing and Monte-Carlo return differences—which requires neither annotations nor domain-specific solution states, and its Bradley-Terry head provides a calibration guarantee that binary step labels do not.

Verifier-Guided Policy Training Verifiers have long guided policy optimization as filters, reward shapers, or rerankers. Rejection sampling and best-of- n selection use outcome labels to curate training data; reward shaping adds dense intermediate rewards to sparse tasks, at the risk of leaving the policy dependent on the shaping signal; and critic-based methods such as PPO (Schulman et al., 2017) fold the judge into the value function itself. STEP-VERIFIER draws on all three roles—filtering, dense feedback, and optional test-time reranking—but differs in the annealing schedule: the feedback weight decays to zero over training, which provably returns the final optimization problem to the outcome-only objective, so the deployed policy needs no verifier and no test-time overhead.

3 STEP-VERIFIER

3.1 Setting and Notation

An agent interacts with an environment over T steps. At step t it receives an environment observation o_t (a tool response, a page state, or an error message) and emits an action span a_t (a tool call, a query, or a composed reply); we write $u_t = (o_t, a_t)$ for the observation-action pair and $h_t = (x, u_{1:t-1})$ for the interaction history preceding step t , where x is the task prompt. A trajectory $\tau = (x, o_1, a_1, \dots, o_T, a_T)$ terminates with a binary outcome reward $R(\tau) \in \{0, 1\}$. The policy π_θ is trained with GRPO (Shao et al., 2024): for each prompt, a group of G rollouts is sampled from the behavior policy, the sequence-level advantage of rollout i is the group-normalized reward $A_i = (R_i - \mu_R) / (\sigma_R + \epsilon)$, and the clipped surrogate with a KL penalty is optimized over the response tokens of every action span. Concrete backbone and environment configurations are reported in Section 4.

3.2 Action-Level Verifier

The verifier is a language model s_θ that scores a single step in context: given the current observation o_t , the candidate action a_t , and a truncated window of the preceding interaction, it outputs a scalar $s_\theta(u_t | h_t)$. The score is relative by construction. Following the Bradley-Terry model (Bradley & Terry, 1952), the probability that action a is judged better than action a' in the same context h is

$$P_\theta(a \succ a' | h) = \sigma(s_\theta(o_h, a | h) - s_\theta(o_h, a' | h)), \quad (1)$$

where o_h is the observation at state h and σ is the logistic function. Two design choices follow from this parametrization. First, the verifier never scores an action in isolation; its output is meaningful only through differences between alternatives in a shared context, which matches how the training comparisons are synthesized (Section 3.3). Second, the score has no absolute scale to learn—only gaps—so the head cannot drift into imitating raw returns and remains invariant to reward-scale changes across tasks.

3.3 Synthesizing Comparisons from Logged Interactions

The verifier’s supervision is distilled from a corpus $\mathcal{D}$ of logged trajectories, each carrying its final outcome label. No fresh environment interaction is used at any point.

Continuation sets. For an interaction history h , let $\mathcal{C}(h) \subseteq \mathcal{D}$ collect the logged trajectories that pass through h ; each element $c \in \mathcal{C}(h)$ is a continuation of h with realized final return $R(c)$. The Monte-Carlo value of h under the logging policy is estimated by

$$\hat{V}(h) = \frac{1}{|\mathcal{C}(h)|} \sum_{c \in \mathcal{C}(h)} R(c). \quad (2)$$

Prefix-split pairs. The comparisons we harvest compare two actions taken from the same state. For a history h and two distinct actions a, a' such that $|\mathcal{C}(h \cdot a)| \geq m$ and $|\mathcal{C}(h \cdot a')| \geq m$ (a minimum support of m logged continuations, guarding against single-sample noise), the two successor states $h \cdot a$ and $h \cdot a'$ share the common prefix h , and their Monte-Carlo return difference

$$\delta(h; a, a') = \hat{V}(h \cdot a) - \hat{V}(h \cdot a') \quad (3)$$

is a pairwise label: if $\delta > 0$, the logged evidence says a was the better choice from h ; if $\delta < 0$, the reverse; ties are discarded. The pair is presented to the verifier as the context h with the two competing observation-action pairs (o_h, a) and (o_h, a') , and the training loss is the Bradley-Terry negative log-likelihood

$$\mathcal{L}_{\text{ver}}(\theta) = -\mathbb{E}_{(h, a, a', y)} [y \log \sigma(\Delta_\theta) + (1 - y) \log \sigma(-\Delta_\theta)], \quad (4)$$

where $\Delta_\theta = s_\theta(o_h, a \mid h) - s_\theta(o_h, a' \mid h)$, and $y = \mathbf{1}[\delta > 0]$ is the synthesized preference. Because both candidates are conditioned on the same context, all context-dependent confounds—task difficulty, prompt formulation, logging-policy quirks up to h —cancel in the comparison, and the signal isolates the consequence of the divergent action itself.

Why prefix sharing suffices. Successful outcome-rewarded training revisits the same tasks many times, and stochastic sampling makes early interaction prefixes recur across attempts; every recurrence multiplies the harvestable pairs. The corpus thus densifies exactly where it matters: heavily visited states, where the policy actually spends its decisions, accumulate large continuation sets and yield many well-supported comparisons, while states visited once contribute nothing and are correctly left unsupervised. Section 4.6 quantifies this supply over training.

3.4 Verifier-in-the-Loop Policy Training

The trained verifier participates in policy optimization through two mechanisms, both of which are switched off before training ends.

Log filtering. Before each policy update, every interaction log in the current batch is scored by the verifier, aggregating step scores into a log-level quality $Q(\tau) = \frac{1}{T_\tau} \sum_t s_\theta(u_t \mid h_t)$; the bottom ρ fraction by quality is removed from the update batch, and the GRPO surrogate is computed over the surviving logs. Filtering targets a failure mode of outcome-rewarded training that no advantage shaping addresses: trajectories whose final reward is noisy or accidental (lucky guesses that corrupt the update with unrepresentative behavior) pollute the batch regardless of how their advantage is scaled.

Annealed dense feedback. For the surviving logs, the verifier supplies intermediate feedback inside the advantage. Step scores are standardized within each batch and folded into the sequence-level advantage at the action-span level,

$$\hat{A}_t = A_{\text{seq}} + \lambda_k \tilde{s}_\theta(u_t \mid h_t), \quad (5)$$

where $\tilde{s}_\theta$ is the batch-standardized score at step t and λ_k is the feedback weight at update k . The weight follows a linear annealing schedule: it starts at λ_0 , decays to zero at a fixed fraction η of the total training budget, and stays at zero afterwards. After the annealing point, Equation 5 reduces to the plain GRPO advantage and the final phase of training is identical to outcome-only optimization; the deployed policy therefore never queries the verifier. The verifier itself is refreshed every K_{ver} updates by continuing its training on the most recent logs, so its judgments track the improving policy rather than the initial one.

3.5 Theoretical Properties

We state two properties of the verifier’s supervision and training objective, with proofs in Appendix A. Together they establish that the synthesized comparisons estimate real value gaps and that the learned scores are calibrated to those gaps.

Proposition 1 (Unbiased and consistent pairwise labels). Condition on the corpus $\mathcal{D}$ and fix a state h with actions a, a' whose logged continuations are exchangeable draws from the logging policy’s continuation distributions with expected returns $Q(h \cdot a)$ and $Q(h \cdot a')$. Then $\hat{V}(h \cdot a)$ is unbiased for $Q(h \cdot a)$, the difference $\delta(h; a, a')$ is unbiased for the value gap $Q(h \cdot a) - Q(h \cdot a')$, and as the continuation counts grow, $\delta(h; a, a')$ converges almost surely to the value gap.

Proposition 2 (Calibration of the Bradley-Terry head). Suppose the synthesized labels follow a Bradley-Terry response model: for a pair with true value gap $\Delta Q = Q(h \cdot a) - Q(h \cdot a')$, the observed preference is $a \succ a'$ with probability $p = \sigma(\Delta Q)$. Then the population risk of

Equation 4 is minimized exactly when the verifier’s score gap equals the value gap, $\Delta_\theta = \Delta Q$, for every harvested pair; on a connected comparison graph the minimizer is unique up to an additive constant, and it recovers the value function itself, $s_\theta = Q + \text{const.}$

Proposition 1 certifies the label source: the return difference between continuation sets sharing a prefix estimates the consequence of the prefix-splitting action, with bias only through the usual off-policy concerns when the logging policy differs from the policy being verified, which the periodic refresh of Section 3.4 mitigates. Proposition 2 explains a phrase in our claims that ordering alone would not license: the verifier’s scores are calibrated—their differences estimate value gaps, not just rankings—which is what allows the dense feedback of Equation 5 to carry quantitative meaning. Two caveats bound the interpretation. The value gaps are those of the logging policy’s continuations, not counterfactual gaps under an improved policy, and the guarantees concern the population objective; finite-support pairs (m small) produce noisy labels whose disagreement with the population optimum decays as m grows. Neither caveat affects the sign consistency that the filtering and feedback roles consume, but both caution against reading a verifier score as an exact action value.

4 Experiments

4.1 Setup

Benchmarks. We evaluate on three agent benchmarks spanning general assistance, tool composition, and multi-turn planning. GAIA (Mialon et al., 2023) tests general assistant abilities over web browsing, file handling, and tool use; we report accuracy on its three validation levels (L1–L3) and the average over all evaluation instances. ToolBench (Qin et al., 2024) requires composing calls to real-world APIs; we report win rate on three representative subsets spanning in-domain single-tool use (I1-Inst), unseen-tool scenarios (I2-Cat), and the hardest unseen-tool-and-instruction split (I3-Inst), with the average over all evaluation instances. AgentBoard (Ma et al., 2024) evaluates multi-turn agents across nine environments; we report its two official metrics, success rate (SR) and progressive rate (PGR).

Baselines. (1) No training: Vanilla evaluates the instruction-tuned backbone directly. (2) Outcome-only RL: GRPO (Shao et al., 2024) trains from final scores alone, and PPO (Schulman et al., 2017) replaces the group baseline with a learned value critic. (3) Verifier-guided RL with matched roles: GRPO + ORM attaches a trajectory-level reward model trained on the same corpus, and GRPO + PRM attaches a process reward model trained with binary step labels derived by thresholding the same Monte-Carlo estimates; both use their scores for log filtering and dense feedback exactly as STEP-VERIFIER does, isolating the verifier’s granularity and label type as the only differences. All RL methods share backbones, environment wrappers, rollout budgets, and schedules.

Implementation. We train Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Mistral-7B-Instruct (Jiang et al., 2023) policies with veRL (Sheng et al., 2024) and vLLM (Kwon et al., 2023) on eight NVIDIA A100 GPUs: AdamW at learning rate 1×10^{-6} , group size $G = 8$, KL coefficient 0.01, clipping radius 0.2, mini-batch of 256 turns. GAIA uses 16 tasks per batch with 4,096/1,024 prompt/response tokens and at most 30 steps; ToolBench and AgentBoard use the same batch size with 2,048/768 tokens and at most 25 and 30 steps, respectively. The log corpus is accumulated from the first third of training (40k trajectories), and the default verifier is Llama-3.2-3B-Instruct (Grattafiori et al., 2024) consuming a 1,024-token interaction window; pairs require minimum support $m = 2$. Defaults: filter fraction $\rho = 20\%$, initial feedback weight $\lambda_0 = 0.1$, annealing point $\eta = 75\%$, refresh interval $K_{\text{ver}} = 50$ updates. The verifier adds 18% wall-clock overhead during training and none at deployment.

4.2 Main Results

Table 1 reports all methods on the three benchmarks under both backbones. Three observations summarize the results.

Table 1: Main results across two backbones. We report accuracy (%) on GAIA, win rate (%) on ToolBench (subsets I1-Inst, I2-Cat, I3-Inst), and success rate and progressive rate (%) on AgentBoard; averages are computed over all evaluation instances. Best per column in bold, second-best underlined.

Method	GAIA				ToolBench				AgentBoard	
	L1	L2	L3	Avg.	I1	I2	I3	Avg.	SR	PGR
Backbone: Llama-3.1-8B-Instruct										
Vanilla	18.5	8.9	2.2	10.4	22.4	14.8	9.2	16.5	11.2	21.6
GRPO	33.7	19.4	6.7	21.5	41.6	27.3	18.5	30.4	24.8	38.9
PPO	31.0	17.2	5.6	19.8	38.2	24.6	16.1	27.9	22.1	35.4
GRPO + ORM	36.2	21.8	<u>8.9</u>	23.2	44.3	29.8	20.4	32.6	27.6	41.8
GRPO + PRM	<u>39.4</u>	<u>24.6</u>	7.8	<u>25.7</u>	<u>46.8</u>	<u>32.1</u>	<u>22.9</u>	<u>34.9</u>	<u>30.2</u>	<u>44.6</u>
STEP-VERIFIER (Ours)	45.1	28.9	13.3	30.1	52.4	38.6	<u>22.5</u>	38.9	35.4	50.3
Backbone: Mistral-7B-Instruct										
Vanilla	15.8	6.7	2.2	8.9	19.6	11.7	7.3	13.9	9.4	18.7
GRPO	30.2	16.4	4.4	18.7	37.8	24.1	15.8	27.2	21.6	35.2
PPO	27.9	14.8	3.3	17.2	34.5	21.6	13.4	24.8	19.8	32.6
GRPO + ORM	32.5	18.2	6.7	20.4	40.2	26.4	17.6	29.3	24.3	37.9
GRPO + PRM	<u>35.8</u>	<u>20.9</u>	<u>10.0</u>	<u>22.6</u>	<u>42.9</u>	<u>28.8</u>	<u>22.4</u>	<u>31.5</u>	<u>26.8</u>	<u>40.4</u>
STEP-VERIFIER (Ours)	41.2	25.4	<u>8.9</u>	26.8	48.6	34.9	<u>21.6</u>	37.1	31.9	46.2

An action-level verifier helps beyond final scores. STEP-VERIFIER outperforms GRPO and PPO on every aggregate metric with both backbones, with the largest margins on GAIA (+8.6 and +8.1 average points over GRPO), the benchmark with the longest and most heterogeneous tool chains. The two verifier-free RL methods land within a point of each other everywhere except AgentBoard, where GRPO’s group baseline edges PPO’s learned critic, a first indication that judging intermediate behavior is a distinct and harder problem than evaluating the whole attempt.

Granularity and label type both matter. The three verifier-guided rows consume identical corpora and play identical roles, differing only in what they score and how they are supervised. Trajectory-level ORM scoring, which cannot localize credit inside an attempt, gains 1.7 points over GRPO on GAIA average (Llama); binary step labels add another 2.5; pairwise Bradley-Terry supervision adds 4.4 more. The ordering is unchanged under the Mistral backbone and on the other benchmarks. The comparison isolates the verifier design itself, since filtering rates, feedback weights, and annealing schedules are matched.

Aggregate wins coexist with narrow losses. STEP-VERIFIER posts the best average on all six benchmark–backbone combinations but concedes individual columns: the binary PRM wins the hardest ToolBench split (I3-Inst) under both backbones and GAIA L3 under Mistral, and the ORM wins GAIA L3 under Llama. The pattern is consistent with the supervision supply: hard, rarely visited states recur less in the log corpus, so their continuation sets stay thin, the minimum support m filters out their pairs, and the verifier is least informed exactly where the baselines’ coarser signals still fire. We return to this supply-side view in Sections 4.6 and 4.4.

4.3 Score Quality Against Human Judgments

Table 2 evaluates the verifier itself against human judgments, independently of any policy training. STEP-VERIFIER agrees with human comparisons on 81.5% of pairs—19 points above the trajectory ORM and 8 above the binary PRM trained on the same Monte-Carlo

Table 2: Verifier quality on held-out logged interactions, scored against human step judgments from three annotators on 2,000 sampled observation–action pairs (inter-annotator agreement $\kappa = 0.77$). Pairwise agreement is the fraction of human-judged comparisons the scorer ranks correctly, Spearman ρ correlates scores with human ratings over all steps, AUROC measures discrimination of steps on trajectories that eventually succeeded versus failed, and ECE is expected calibration error of score gaps against human-judged gaps.

Scoring model	Pairwise agr. (%)	Spearman ρ	AUROC	ECE
Trajectory ORM	62.4	0.41	68.2	0.19
MC value regression	68.7	0.48	72.5	0.14
Binary PRM	73.2	0.56	76.8	0.11
STEP-VERIFIER	81.5	0.67	83.4	0.06

Table 3: Ablations with Llama-3.1-8B-Instruct. Each row disables or degrades one design decision under the matched schedule and evaluation protocol.

Variant	GAIA Avg.	ToolBench Avg.	AgentBoard SR
STEP-VERIFIER (full)	30.1	38.9	35.4
Verifier supervision			
Binary labels (no Bradley-Terry head)	25.7	34.9	30.2
Random pairs (no common-prefix constraint)	27.9	36.0	32.9
In-the-loop mechanisms			
w/o log filtering	27.4	36.4	32.6
w/o dense feedback	28.3	37.2	33.2
w/o feedback annealing (constant weight)	26.5	35.1	31.4
Verifier maintenance			
Frozen verifier (no refresh)	28.6	37.0	32.7

material—and its Spearman correlation with human step ratings (0.67) is two-thirds higher than the ORM’s. The calibration column is the direct empirical counterpart of Proposition 2: its score gaps err against human-judged gaps with expected calibration error 0.06, roughly a third of the ORM’s, supporting the use of score differences as quantitative feedback in Equation 5 rather than as a bare ranking. MC value regression, which predicts absolute returns without the shared-context comparison, trails the Bradley-Terry head by 13 points of pairwise agreement despite seeing the same continuation sets, confirming that context-conditioned comparison, not the underlying estimate, is where the alignment is won.

4.4 Ablations

Table 3 removes one design decision at a time. The largest single degradation comes from replacing the Bradley-Terry head with binary labels, which drops the system to the GRPO + PRM row of Table 1 and confirms that calibration, not merely correctness of individual labels, carries the policy gains. Relaxing the common-prefix constraint to random state pairs costs 2–3 points on every benchmark: without shared context, comparisons must generalize across confounds the prefix constraint cancels, and the learned scores blur. Among the in-the-loop mechanisms, annealing is the most critical—keeping the feedback weight constant costs 3.6 GAIA points and finishes below the filtering-only variant, showing that a verifier signal that never exits actively harms late training as the policy surpasses the logging distribution—while filtering and feedback each contribute 2–3 points independently. Freezing the verifier after initial training costs 1.5–2.7 points, a milder but consistent penalty consistent with its supervision drifting stale as the policy improves.

4.5 Hyperparameter Sensitivity

Table 4 varies the four principal hyperparameters one at a time. Every grid is peaked at or near the default, and each shape admits a mechanism-level reading. The feedback weight λ_0 is benign up to 0.2 and degrades sharply at 0.4, where verifier scores begin to

Table 4: Hyperparameter sensitivity with Llama-3.1-8B-Instruct, varying the initial feedback weight λ_0 , the filter fraction ρ , the annealing point η , and the verifier capacity one at a time around the defaults.

Hyperparameter	Setting	GAIA Avg.	ToolBench Avg.	AgentBoard SR
Initial feedback weight λ_0				
	0	28.3	37.2	33.2
	0.05	29.4	38.1	34.5
	0.1 (default)	30.1	38.9	35.4
	0.2	29.7	38.6	35.0
	0.4	27.8	36.3	33.0
Filter fraction ρ				
	0%	27.4	36.4	32.6
	10%	29.3	37.9	34.3
	20% (default)	30.1	38.9	35.4
	30%	28.8	37.5	34.1
Annealing point η				
	constant (no annealing)	26.5	35.1	31.4
	50%	29.0	37.8	34.0
	75% (default)	30.1	38.9	35.4
	100%	29.2	38.0	34.3
Verifier capacity				
	Llama-3.2-1B-Instruct	28.5	36.9	33.4
	Llama-3.2-3B-Instruct (default)	30.1	38.9	35.4
	Llama-3.1-8B-Instruct	30.3	39.1	35.6

Table 5: Verifier and feedback statistics over the early (0–33%), middle (33–66%), and late (66–100%) phases of policy training with Llama-3.1-8B-Instruct.

Statistic	Early	Middle	Late
Verifier pairwise agreement with humans (%)	76.9	80.4	82.7
Verifier AUROC (eventual success)	78.1	82.6	84.3
Mean feedback weight λ_k ($\times 10^{-2}$)	10.0	4.2	0.0
Filtered logs that ended in failure (%)	71.4	74.8	76.9

dominate the outcome-derived advantage and the policy optimizes the judge rather than the task. The filter fraction ρ trades coverage against purity: filtering nothing forfeits the mechanism’s gain, while filtering 30% discards genuinely informative borderline logs along with the noise. The annealing point η shows the same profile as the corresponding ablation—any constant-weight alternative is the worst setting in the grid, and annealing too early (50%) wastes verifier capacity that is still well-calibrated, while annealing exactly at the end (100%) leaves no verifier-free consolidation phase. Verifier capacity saturates between 1B and 3B; doubling again to 8B moves results by at most 0.2 points, indicating that the binding constraint is the supervision supply, not the scorer’s capacity.

4.6 Training Dynamics and Deployment Independence

Table 5 tracks the verifier’s lifecycle during policy training. Its judgments improve monotonically—pairwise agreement with humans rises from 76.9% to 82.7% and success-prediction AUROC from 78.1 to 84.3—because the refreshed corpus keeps supplying pairs from states the current policy actually visits, and heavily visited states accumulate ever-larger continuation sets. The filtering mechanism sharpens in parallel: the share of filtered logs that ended in failure grows from 71.4% to 76.9%, meaning the discarded 20% of each batch is increasingly the right 20%. Meanwhile the feedback weight decays to zero by construction, so the verifier’s improving scores are spent where they are cheapest—early, when the outcome signal is noisiest—and withdrawn as the policy’s own outcomes become informative.

Table 6: Deployment independence with Llama-3.1-8B-Instruct: final-policy evaluation with and without the verifier at test time.

Deployment mode	GAIA Avg.	ToolBench Avg.	AgentBoard SR
Policy alone (default)	30.1	38.9	35.4
Policy + test-time verifier filtering	30.0	38.8	35.3
Policy + best-of-4 verifier reranking	31.4	39.8	36.1

Table 6 verifies the deployment claim. The final policy evaluated alone matches its training-time numbers exactly, and attaching the verifier at test time changes nothing (≤ 0.1 points under filtering): the annealed schedule leaves no residual dependency. Best-of-4 reranking recovers only about one point on GAIA and less elsewhere, a small optional gain that is an order of magnitude below the training-time improvement, confirming that the verifier’s value was realized during optimization rather than deferred to inference.

5 Conclusion

This paper asked whether the quality of individual agent actions can be judged before an episode ends, without human step labels or extra environment interaction. STEP-VERIFIER answers by distilling a success-labeled log corpus into action-level comparisons: Monte-Carlo return differences between states sharing a common prefix supply unbiased pairwise supervision, a Bradley-Terry head converts the pairs into calibrated per-action scores, and the verifier spends a bounded training-time tenure filtering logs and densifying feedback before annealing out entirely, leaving a policy that deploys alone. Across GAIA, ToolBench, and AgentBoard with two backbones, the approach outperforms outcome-only training and both trajectory-level and binary-label verifier baselines, and its scores align with human judgments substantially better than either. The supply-side analysis also traces the method’s boundary: supervision exists only where logged continuations recur, so the hardest, rarely visited states are exactly where the verifier is least informed. Synthesizing comparisons for those states—through targeted exploration that revisits uncertain prefixes, or through cross-task transfer of verifier scores—is the natural continuation of this work.

AI use statement

Generative AI tools supported literature organization and language editing during manuscript preparation. The authors verified all AI-assisted passages and assume full responsibility for the paper, including the soundness of the claims, the accuracy of the references, and the correspondence between reported numbers and the experiments conducted.

Ethics statement

This research trains and evaluates language-model agents in simulated research environments on publicly released benchmarks. No human subjects or private data are involved beyond voluntary annotation of logged interactions by the authors’ institution. Deploying the resulting policies or verifiers in production systems would require application-specific safety review, access control, and monitoring that fall outside the scope of this study.

Reproducibility statement

Section 4 documents the benchmarks, baseline configurations, and hyperparameters; Appendix A contains complete proofs of Propositions 1 and 2. The supplementary material provides the corpus-construction pipeline, pair-synthesis code, verifier training scripts, and policy training configurations sufficient to regenerate all reported numbers.

References

- Ralph A. Bradley and Milton E. Terry. Rank analysis of incomplete block designs I: The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.
- Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025.
- Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.
- 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, 2026.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 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.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 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.
- Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.
- Zhaorui Ma, Yidong Yang, Yuguang Zhou, Jiaheng Wei, Xiwen Chen, Cheng Zhu, Songze Huang, Xinyu Zhang, Shufei Wang, et al. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1934–1964, 2024.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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

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

Guangbin Sheng, Chi Cui, Xiaoyu Zhang, Kaixiang Diao, Yifei Wu, Yongxiang Lin, Jiahui Zhang, Hang Yu, Conghui Zhu, et al. Hybridflow: A flexible and efficient RLHF framework. In Proceedings of the 29th European Conference on Computer Vision (ECCV), 2024.

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, 2025.

Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. Advances in Neural Information Processing Systems, 38:113222–113244, 2026.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. arXiv preprint arXiv:2402.19446, 2024.

A Proofs

This appendix proves Propositions 1 and 2.

A.1 Proof of Proposition 1

Proof. Fix a state h and an action a , and let $\mathcal{C}(h \cdot a) = \{c_1, \dots, c_n\}$ be the logged continuations from $h \cdot a$, each realized with a final return $R(c_j)$. Under the assumption that these continuations are exchangeable draws from the logging policy’s continuation distribution at $h \cdot a$, each $R(c_j)$ has expectation $Q(h \cdot a)$, the expected return of the logging policy from the successor state. The Monte-Carlo estimate is the sample mean

$$\hat{V}(h \cdot a) = \frac{1}{n} \sum_{j=1}^n R(c_j), \quad (6)$$

so by linearity of expectation,

$$\mathbb{E}[\hat{V}(h \cdot a)] = \frac{1}{n} \sum_{j=1}^n \mathbb{E}[R(c_j)] = Q(h \cdot a), \quad (7)$$

which establishes unbiasedness of the per-state estimate. For two actions a, a' with independent continuation sets of sizes n and n' , the difference satisfies

$$\mathbb{E}[\delta(h; a, a')] = Q(h \cdot a) - Q(h \cdot a'), \quad (8)$$

again by linearity, so the pairwise label’s underlying estimate is unbiased for the value gap.

For consistency, fix a state-action pair and let $n \rightarrow \infty$ with $R(c_1), R(c_2), \dots$ i.i.d. (the exchangeability assumption) and bounded (terminal returns are binary). By the strong law of large numbers,

$$\hat{V}(h \cdot a) = \frac{1}{n} \sum_{j=1}^n R(c_j) \xrightarrow{\text{a.s.}} \mathbb{E}[R(c)] = Q(h \cdot a). \quad (9)$$

Since almost-sure convergence is preserved under differences of independent sequences,

$$\delta(h; a, a') \xrightarrow{\text{a.s.}} Q(h \cdot a) - Q(h \cdot a'), \quad (10)$$

which is the stated convergence. The induced binary preference $\mathbf{1}[\delta > 0]$ then agrees almost surely with the sign of the true value gap in the limit; for finite support, $\Pr[\mathbf{1}[\delta > 0] \neq \mathbf{1}[\Delta Q > 0]]$ is controlled by the variance of δ , which is $\text{Var}(R) \left(\frac{1}{n} + \frac{1}{n'} \right)$ and decays as the support grows—the quantitative counterpart of the minimum-support filter m .

A.2 Proof of Proposition 2

Proof. Consider a single harvested pair with true value gap $\Delta Q = Q(h \cdot a) - Q(h \cdot a')$, and suppose the observed preference follows the Bradley-Terry response model: $a \succ a'$ with probability $p = \sigma(\Delta Q)$, independently across pairs. Let $g = \Delta_\theta = s_\theta(o_h, a \mid h) - s_\theta(o_h, a' \mid h)$ denote the verifier’s score gap on this pair. The population risk of the pair’s contribution to Equation 4 is

$$f(g) = -p \log \sigma(g) - (1 - p) \log \sigma(-g). \quad (11)$$

Differentiating and using $\sigma'(g) = \sigma(g)(1 - \sigma(g))$,

$$f'(g) = -p \frac{\sigma(g)(1 - \sigma(g))}{\sigma(g)} + (1 - p) \frac{\sigma(g)(1 - \sigma(g))}{1 - \sigma(g)} = \sigma(g) - p, \quad (12)$$

and

$$f''(g) = \sigma(g)(1 - \sigma(g)) > 0, \quad (13)$$

so f is strictly convex with a unique stationary point. Setting $f'(g) = 0$ gives $\sigma(g) = p = \sigma(\Delta Q)$, and since σ is injective,

$$g^* = \Delta Q. \quad (14)$$

The per-pair population risk is therefore minimized exactly when the verifier’s score gap equals the true value gap, and nowhere else.

The full population risk is a weighted sum of such terms over all harvested pairs. Because every per-pair minimizer is simultaneously attainable—assigning the verifier the potential $s_\theta = Q + \text{const}$ realizes $g = \Delta Q$ on every pair, since value gaps are differences of a common potential—the sum is minimized by any scorer whose gaps equal the true gaps pairwise. Conversely, any deviation on some pair strictly increases that pair’s strictly convex term, so the minimizer satisfies the gap-equality on all pairs. If the comparison graph (nodes are states, edges are harvested pairs) is connected, the equalities $\Delta_\theta = \Delta Q$ over edges determine the scores up to a single additive constant, giving uniqueness in the stated sense and $s_\theta = Q + \text{const}$ as the exact population optimum. This is the calibration property: score gaps estimate value gaps, not merely their signs.