LLM VALUE FUNCTIONS FOR CREDIT ASSIGNMENT IN AGENTIC RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) for large language model (LLM) agents remains challenging in long-horizon tasks with sparse outcome rewards, where assigning credit to individual actions is difficult. Methods such as GRPO assign the same trajectory-level advantage to all actions, while critic-based methods require learning an additional value function that can be costly and unstable. We introduce LLM Value Function RL (LVRL), which uses the pretrained knowledge and in-context learning ability of an LLM to estimate policy-aware state values without finetuning a separate critic. LVRL conditions the value model on group context and calibrates its predictions using known values at the initial and terminal states, enabling step-level credit assignment from estimated value differences. We further extend this idea to learning from an expert policy through on-policy distillation (OPD), introducing LLM Value Function OPD (LVOPD), which focuses teacher supervision on actions associated with decreases in estimated value. We evaluate our approach across diverse agentic benchmarks: AppWorld, BabyAI, and SWE-bench Verified. Across policy scales (4B to 32B), LVRL consistently and substantially outperforms both GRPO and PPO. On AppWorld and SWE-bench Verified, LVRL yields absolute improvements of up to 13.6 and 8.6 percentage points over GRPO. Notably, on tasks requiring extended exploration like BabyAI, LVRL bridges the gap from total failure to 75.0% success. Furthermore, LVOPD yields consistent improvements over standard distillation baselines on AppWorld. Finally, Monte-Carlo analysis confirms that our frozen critic accurately identifies semantic pitfalls and recoveries, proving that LVRL extracts robust, policy-aware step-level credit from sparse terminal outcomes.

1 INTRODUCTION

Language models are increasingly deployed as agents that reason and act over dozens of steps to solve complex tasks (Yao et al., 2022; Yang et al., 2024; Jimenez et al., 2024; Trivedi et al., 2024). Reinforcement learning allows these agents to improve from experience, but environment rewards are typically sparse and outcome-based: a single binary signal at the end of the episode indicating success or failure. Distributing this single bit of feedback across tens of individual actions is the classical problem of temporal credit assignment (Sutton & Barto, 1998).

Current policy optimization methods face a trade-off between coarse trajectory-level credit and the high cost of learning a value function. Outcome-only methods like GRPO (Shao et al., 2024) estimate a group-relative baseline and assign a scalar advantage to *every* action in a trajectory. This uniform signal fails to distinguish individual contributions: mistakes in successful trajectories receive positive credit, while useful actions in failed trajectories receive negative credit. Alternatively, methods like PPO (Schulman et al., 2017) learn a value function to estimate step-level advantages. However, training a critic adds computational overhead and struggles to accurately evaluate long interaction histories from sparse rewards (Kazemnejad et al., 2025). While recent work explores alternative step-level credit mechanisms, such as process reward models (Lightman et al., 2024), Monte-Carlo continuations (Kazemnejad et al., 2025), or outcome-conditioned heuristics (Tan et al., 2026), they often require human annotations, extra rollout budgets, or rely on proxy metrics.

To overcome these limitations, we introduce LLM Value Function RL (LVRL), a method that extracts dense, per-step credit using a strong, frozen LLM as a value critic. By evaluating completed

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

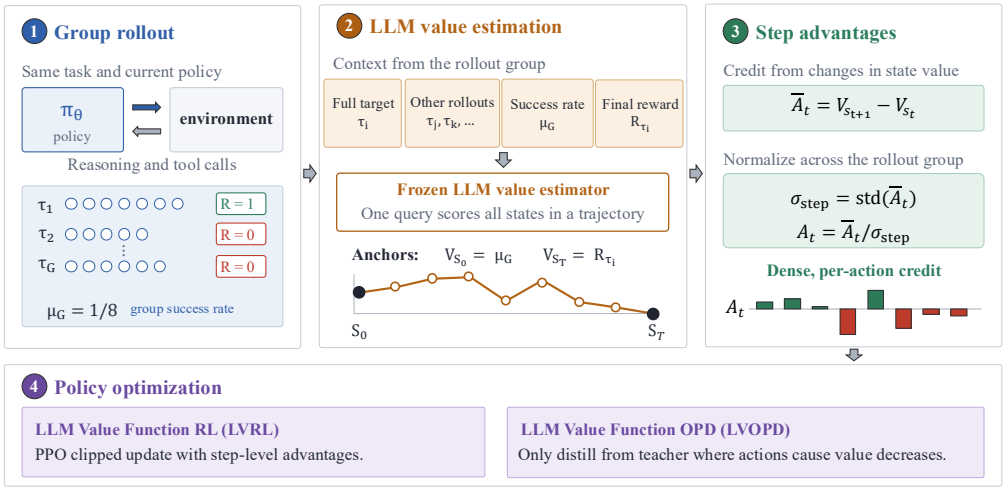


Figure 1: Overview of the proposed framework. (**1**) **Group rollout:** We sample a group of trajectories and their sparse outcome rewards from the current policy to compute the empirical group success rate μ_G. (**2**) **LLM value estimation:** A frozen LLM critic scores all trajectory states, conditioned on group context. Estimates are strictly anchored to empirical outcomes: the initial state to the group success rate and the terminal state to the actual reward. (**3**) **Step advantages:** We compute unnormalized step advantages as consecutive value differences ($\overline{A}_t = V_{S_{t+1}} - V_{S_t}$) and normalize them across the group, yielding a dense, per-action credit signal. (**4**) **Policy optimization:** These step-level advantages drive standard PPO updates in LVRL, or act as a gating mechanism in LVOPD to restrict teacher supervision to actions that decrease the estimated value.

trajectories in hindsight, the frozen LLM critic observes downstream consequences to accurately assess earlier actions, such as ignored tool errors or successful recoveries. Because value estimation depends on the acting policy, we make the frozen LLM *policy-aware* through in-context learning. Specifically, we condition the estimator on summaries of other rollouts from the same training group, along with their empirical success rate and final rewards. This dynamically updates the critic's context with the current policy's behavior and task performance, without requiring any parameter updates. We then convert the predicted values into step-level advantages via consecutive value differences (Ng et al., 1999). Crucially, by anchoring the initial state to the group success rate and the terminal state to the observed final reward, we guarantee that these step-level advantages sum exactly to the trajectory's overall group-relative return. The frozen critic thus acts as a precise redistributor of sparse outcome rewards across individual actions.

We further extend this step-level signal to on-policy distillation (OPD), where a stronger teacher provides token-level supervision (Agarwal et al., 2024; Gu et al., 2024). Because uniform distillation penalizes large KL divergence everywhere, it can suppress alternative, successful strategies discovered by the student. To align distillation with policy optimization, LVOPD introduces *value-guided gating*. By restricting teacher supervision strictly to actions that decrease the estimated value, we selectively correct execution errors while preserving the student's independent progress.

We train and evaluate LVRL and LVOPD on AppWorld, BabyAI, and SWE-bench Verified across policy scales ranging from 4B to 32B parameters. In LVRL, we utilize Claude Opus 4.6 as the frozen LLM value critic. It consistently improves final generalization over outcome-only supervision (GRPO) and learned value functions (PPO). For instance, on Qwen3-8B, LVRL achieves 26.3% success on AppWorld and 20.2% on SWE-bench Verified, compared to 19.8% and 12.0% for GRPO, respectively. Notably, LVRL is particularly critical in environments requiring extended exploration: on BabyAI environments, it reaches 65.5% success on out-of-distribution scenarios, whereas baseline methods fail entirely. Furthermore, LVOPD yields consistent improvements over standard distillation baselines on AppWorld, reaching a 20.2% success rate with Qwen3-4B and improving pure OPD by 3.2 percentage points.

Finally, we validate the accuracy of the frozen LLM critic's value predictions. Using Monte-Carlo rollouts as a ground-truth probe, we demonstrate that predicted value changes correctly identify 72% of critical pitfalls and 66% of recoveries. Furthermore, our ablations highlight two critical insights:

in-context policy calibration is essential for maximizing improvements, and the method's overall efficacy is inherently bounded by the reasoning capacity of the chosen LLM critic. In summary, we present a unified framework that derives a dense, per-step credit signal entirely from a frozen, policy-aware LLM, guiding both RL and selective distillation without any critic updates.

2 Related Work

Step-level credit assignment for LLM agents. Classical actor–critic methods address temporal credit assignment by learning a parameterized value function (Schulman et al., 2015; 2017), an approach extended to multi-turn language agents by ArCHer (Zhou et al., 2024) and GenAC (Shan et al., 2026). To avoid the overhead of critic optimization, recent methods explore alternative credit mechanisms. The GRPO family of algorithms (Shao et al., 2024; Liu et al., 2025; Yu et al., 2025) relies on group-relative trajectory returns, uniformly assigning the same advantage to all actions within a rollout. Another line of work relies on extensive Monte-Carlo sampling; VinePPO (Kazemnejad et al., 2025) explicitly rolls out continuations from intermediate states to estimate state values, whereas GiGPO (Feng et al., 2025) and SALT (Li et al., 2026a) achieve a similar effect implicitly by aggregating returns over shared states across multiple trajectories.

Other approaches rely on auxiliary models or proxy signals. Process reward models are trained with intermediate supervision (Lightman et al., 2024; Zhou et al., 2025; Wang et al., 2025b; Xi et al., 2026; Wu et al., 2026), while implicit estimators derive step- or token-level credit from hindsight judgments of the policy LLM (Tan et al., 2026), target-answer likelihoods (Wang et al., 2025a; Tao et al., 2026; Oh et al., 2026), or internal signals of the policy, such as the discounted KL divergence of subsequent tokens (Ma et al., 2026), policy-to-reference log-ratios (Kang et al., 2026), and attention patterns (Li et al., 2025). Since these implicit estimators are tailored for single-turn reasoning with static answers, adapting them to multi-turn agentic tasks would require non-trivial modifications to handle dynamic environment feedback, complicating direct comparisons. Conceptually related to our approach, GVL (Ma et al., 2025) uses a frozen vision-language model to estimate task progress from robot videos via in-context learning. However, GVL produces policy-agnostic progress scores. LVRL conditions the LLM on rollouts from the current policy and explicitly anchors its predictions to observed outcomes. This formulation ensures that the estimated values are both dynamically policy-aware and consistent with actual environment returns.

On-policy distillation. Knowledge distillation transfers capabilities from a stronger teacher to a student model (Hinton et al., 2015). For generative tasks, on-policy distillation (OPD) mitigates exposure bias by evaluating the teacher on prefixes generated by the student (Agarwal et al., 2024), often optimizing a reverse-KL objective (Gu et al., 2024). However, standard token-level distillation is overly rigid, penalizing deviation from the teacher even when the student discovers a valid alternative strategy. Recent methods adapt teacher-guided updates using token- or step-level likelihood differences (Jia et al., 2026; Zhong et al., 2026), verifier rewards combined with teacher signals (Akhondzadeh et al., 2026; Yang et al., 2026; Xu et al., 2026; Lin et al., 2026), or environment-informed teacher signals (Zhou et al., 2026; Li et al., 2026b). SAGE-OPD explicitly allows valid alternative actions to skip supervision (Zhou et al., 2026), while FutureBridge-OPD validates candidate teacher interventions through student continuations scored using teacher-relative preferences (Chen et al., 2026). These approaches typically rely on specialized heuristics or complex combinations of environment and teacher signals to define their gating rules. Our approach, LVOPD, offers a unified and streamlined alternative: it weights reverse-KL distillation strictly by the negative part of consecutive value differences estimated by the frozen LLM critic. Actions with non-negative predicted value changes therefore receive no distillation loss, successfully preserving the student's valid alternative strategies even when they differ from the teacher's preferred behavior.

3 Method

3.1 Preliminaries

We model agentic tasks as finite-horizon MDPs over interaction histories. At environment step t, the state s_t contains the goal g and the history of observations and actions preceding action a_t. The LLM policy samples $a_t \sim \pi_\theta(\cdot \mid s_t)$ as a sequence of tokens. A trajectory

$\tau = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T, r_T)$ receives only a binary terminal reward $R(\tau) = r_T \in \{0, 1\}$, indicating task success. With discount factor $\gamma = 1$, the objective is to maximize $\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$. For $t < T$, the value function $V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi}[R(\tau) \mid s_t]$ represents the probability of eventual task success. Throughout, t indexes environment steps rather than individual tokens.

PPO (Schulman et al., 2017) learns a value estimator $\hat{V}_\psi(s_t)$ and computes advantages $\hat{A}_t$ from observed rewards and value estimates using Generalized Advantage Estimation (GAE) (Schulman et al., 2015). For trajectories sampled from $\pi_{\theta_{\text{old}}}$, let $r_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ denote the likelihood ratio. PPO maximizes the clipped surrogate:

$$\mathcal{J}_{\text{PPO}}(\theta) = \widehat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{1}$$

where $\epsilon > 0$ is the clipping coefficient and $\widehat{\mathbb{E}}_t$ denotes an empirical average over sampled steps.

To avoid learning a critic, GRPO (Shao et al., 2024) computes a baseline from a group of G trajectories $\{\tau_1, \ldots, \tau_G\}$ sampled from $\pi_{\theta_{\text{old}}}$ for the same task. It assigns a uniform advantage $\hat{A}_i = (R(\tau_i) - \mu_G)/\sigma_G$ to *every* action in τ_i, where μ_G and σ_G are the group mean and standard deviation of the rewards. In multi-step tasks, this uniform credit inadvertently reinforces mistakes in successful trajectories and penalizes optimal actions in failed trajectories.

3.2 LLM Value Function RL

LLM Value Function RL (LVRL) uses a frozen LLM to estimate state values from completed trajectories and converts these estimates into step-level advantages without requiring critic updates. The overall pipeline is illustrated in Figure 1.

The core of our approach lies in utilizing the frozen LLM, denoted as V_ϕ, to generate step-level value estimates. For each task, we sample a group of trajectories $\mathcal{G} = \{\tau_1, \ldots, \tau_G\}$ from the current policy and compute their mean return μ_G. For each target trajectory τ_i, we randomly select k other trajectories from $\mathcal{G}$ as context. A single query then jointly estimates all interior state values as $V_\phi(s_{i,t} \mid \tau_i, \{S(\tau_j)\}_{j\in\mathcal{C}_i}, \mu_G)$, where $\mathcal{C}_i \subseteq \{1, \ldots, G\} \setminus \{i\}$. To reduce context length, the summarization function $S(\cdot)$ summarizes the essential action–observation history of the rollouts (see Appendix C). We anchor the endpoints to empirical data: the initial state to the group success rate ($V_\phi(s_{i,0}) = \mu_G$) and the terminal state to the observed final reward ($V_\phi(s_{i,T_i}) = R(\tau_i)$).

Inspired by potential-based shaping (Ng et al., 1999), we define the unnormalized step-level advantage as $\overline{A}_{i,t} = V_\phi(s_{i,t+1}) - V_\phi(s_{i,t})$. Since intermediate environment rewards are zero, setting $\gamma = 1$ makes this formulation mathematically equivalent to Generalized Advantage Estimation (GAE) at $\lambda = 0$. However, V_ϕ is conditioned on the completed trajectory and its outcome, so it is a hindsight estimate rather than a real state-value function. Let σ_{step} denote the standard deviation of these unnormalized advantages across all valid environment steps in the entire rollout group $\mathcal{G}$, computed over the set $\{\overline{A}_{j,t} \mid j \in \{1, \ldots, G\}, 0 \le t < T_j\}$. We compute the final normalized step advantage as $A_{i,t} = \overline{A}_{i,t}/\sigma_{\text{step}}$. LVRL then replaces the learned advantage $\hat{A}_t$ in Equation (1) with our normalized step-level advantage $A_{i,t}$. While $A_{i,t}$ is computed at the environment-step level, the policy generates each action $a_{i,t}$ as a sequence of tokens. Therefore, all tokens within $a_{i,t}$ inherit the exact same $A_{i,t}$, whereas the likelihood ratio and clipping are evaluated per-token. The final policy loss $\mathcal{L}_{\text{LVRL}}(\theta; \mathcal{B})$ is the negative empirical mean of this token-level surrogate across the minibatch.

The endpoint anchors mathematically guarantee that the unnormalized advantages telescope: $\sum_{t=0}^{T_i-1} \overline{A}_{i,t} = R(\tau_i) - \mu_G$, matching the numerator of the GRPO advantage. The normalized advantages satisfy $\sum_{t=0}^{T_i-1} A_{i,t} = (R(\tau_i) - \mu_G)/\sigma_{\text{step}}$. The estimator thus redistributes an outcome-grounded, trajectory-level signal across environment steps while preserving its unnormalized sum. Note that process reward models (PRMs) inject *learned additive reward* into the RL objective, allowing the policy to exploit estimation errors. For example, the policy can exploit these errors to accumulate a positive total reward on a trajectory that fails. Because the per-step advantages obtained by LVRL sum to the empirical outcome advantage, the frozen critic cannot inject new reward; it only redistributes it. Estimator error is strictly confined to *within-trajectory* credit assignment and cannot flip the sign of a trajectory's overall advantage.

3.3 LLM VALUE FUNCTION OPD

LVRL uses the step-level advantage to determine how sampled behavior contributes to the policy update. When a stronger teacher model with accessible token distributions is available, this same advantage can serve a complementary role: determining *where* teacher supervision should be applied. We refer to this joint setting as ***LLM Value Function OPD*** (LVOPD).

Standard OPD trains the student on its own generated prefixes by minimizing the divergence against token-level targets from a frozen teacher π_E. However, applying this supervision uniformly throughout a trajectory is problematic. A large KL divergence does not necessarily imply that the student's behavior is incorrect; the student might be employing a different, yet effective, reasoning strategy or surface form (e.g., exploring a software repository in a different but valid order). Forcing the student to match the teacher everywhere can suppress alternative behaviors that contribute to task success.

The step-level advantage from Section 3.2 provides a principled mechanism to disentangle task failure from benign divergence. When $A_{i,t} \geq 0$, the sampled action is estimated to contribute positively or neutrally to the outcome. In this scenario, distillation is either redundant (if the student and teacher align) or potentially counterproductive (if they diverge), as uniform supervision risks penalizing the student's valid alternative strategies. Conversely, when $A_{i,t} < 0$, the action reduces the estimated task value. On these steps, the RL policy gradient merely pushes the policy *away* from the mistake, whereas distillation supplies a stronger reference distribution to guide the student toward a correct recovery path. Consequently, teacher supervision is most critical on steps with negative advantages, motivating our selective gating mechanism.

We implement this value-guided distillation by using the step-level advantage to gate the reverse-KL objective. For trajectory τ_i, let $a_{i,t} = (a_{i,t,1}, \ldots, a_{i,t,L_{i,t}})$ denote the action taken at step t with length $L_{i,t}$, and let $h_{i,t,u} = (s_{i,t}, a_{i,t,<u})$ denote the prefix preceding token $a_{i,t,u}$. We define the gating function as $g(A_{i,t}) = \max(-A_{i,t}, 0)$. This gate strictly activates on actions that decrease the estimated task value, dynamically scaling the distillation weight by the magnitude of the predicted drop. The resulting value-guided distillation loss is:

$$\mathcal{L}_{\mathrm{VD}}(\theta; \mathcal{B}) = \frac{1}{N_{\mathcal{B}}} \sum_{\tau_i \in \mathcal{B}} \sum_{t=0}^{T_i - 1} \sum_{u=1}^{L_{i,t}} g(A_{i,t}) \, \mathrm{KL}(\pi_\theta(\cdot \mid h_{i,t,u}) \,\|\, \pi_E(\cdot \mid h_{i,t,u})) \,, \tag{2}$$

where $N_{\mathcal{B}} = \sum_{\tau_i \in \mathcal{B}} \sum_{t=0}^{T_i - 1} L_{i,t}$ is the total number of valid action tokens in the minibatch $\mathcal{B}$. For computational efficiency, we implement this KL divergence using a renormalized top-k approximation over the vocabulary, details can be found in Appendix B.3. Crucially, while the gate $g(A_{i,t})$ is defined at the step level and shared across all tokens of action $a_{i,t}$, the KL divergence is computed over the vocabulary distribution at each token step. LVOPD jointly optimizes the clipped policy surrogate and this gated distillation loss: $\mathcal{L}_{\mathrm{LVOPD}}(\theta; \mathcal{B}) = \mathcal{L}_{\mathrm{LVRL}}(\theta; \mathcal{B}) + \beta \, \mathcal{L}_{\mathrm{VD}}(\theta; \mathcal{B})$, where $\beta \geq 0$ controls the distillation strength.

4 EXPERIMENTS

4.1 SETUP

Benchmark environments. We train and evaluate on three multi-step agent environments spanning distinct capabilities: digital tool use, spatial navigation, and repository-level software engineering. **AppWorld** (Trivedi et al., 2024) requires an agent to complete everyday digital tasks by writing and executing code against a suite of simulated applications. We use the 90 training tasks during learning. **BabyAI** (Chevalier-Boisvert et al., 2018) provides text navigation and interaction tasks in a grid world. To rigorously test sustained multi-step decision-making, we use a modified Pickup benchmark arranged in a 2×3 grid of six rooms. This topology requires the agent to navigate across rooms, open doors, and collect objects based on egocentric textual observations and low-level actions. We train on 285 grid configurations with randomized object positions and target door distances in $\{1, 2, 3\}$. The policy receives textual egocentric observations and emits one low-level action per step, like move forward or turn left. **R2E-Gym** (Jain et al., 2025) provides repository-level software-engineering tasks in which the agent interacts with the codebase through the four tools of the R2E-Gym scaffold (`search`, `file_editor`, `execute_bash`, and `finish`). We use $3,200$ R2E-Gym tasks and the R2E-Gym agent scaffold for training. All three environments provide sparse

outcome rewards, with task success determined at the end of each trajectory. Full details about these training environments and the following evaluation environments are in Appendix A.

Training protocol and baselines. We fine-tune Qwen3-4B and Qwen3-8B on AppWorld and BabyAI, and Qwen3-8B and Qwen3-32B on R2E-Gym (Yang et al., 2025). All policies operate in non-thinking mode to isolate the effect of RL optimization. We run three training seeds on AppWorld and BabyAI and one seed on R2E-Gym due to the higher computational cost of this environment. All methods use a group size of $G = 8$. Full training details are in Appendix B.

For LVRL, we compare against GRPO, which estimates advantages without learning a value function, and PPO, which learns a parameterized value function for advantage estimation. We use Claude Opus 4.6 (Anthropic, 2026) as the frozen LLM value critic with three in-context examples. The estimated API costs per trajectory for each environment are summarized in Appendix B.1.

For LVOPD, we mainly focus our evaluation on AppWorld, comparing against two distillation baselines to isolate the impact of step-level gating. *Pure OPD* applies teacher supervision uniformly across all generated tokens. *Trajectory-gated OPD* replaces both components of LVOPD with their trajectory-level counterparts: it optimizes the GRPO objective and gates the same reverse-KL loss with the GRPO advantage. Consequently, it applies a single, uniform supervision weight to the entire rollout, penalizing or rewarding all steps equally. For our LVOPD implementation, Qwen3-235B-A22B-Thinking-2507-FP8 (Qwen3-235B for brevity) serves as both the distillation teacher and the frozen LLM value critic, with the distillation strength set to $\beta = 1$.

Evaluation protocol. We evaluate AppWorld on the Test Challenge split and R2E-Gym on SWE-Bench Verified (Jimenez et al., 2024). For BabyAI, we evaluate on 300 in-distribution layouts (door distances $\in \{1, 2, 3\}$) and 200 out-of-distribution layouts (distances $\in \{4, 5\}$). For all environments, we test the final training checkpoint three times. For AppWorld and BabyAI, we report the mean and standard deviation across the three training seeds. For R2E-Gym, we report the mean and standard deviation across the three evaluation runs of the single seed.

4.2 Does LVRL Improve Long-Horizon Agent Learning?

We evaluate whether LVRL translates to superior generalization on held-out tasks. As shown in Table 1, LVRL consistently outperforms GRPO and PPO across all domains and model scales. On AppWorld, LVRL achieves a success rate of 24.3% with Qwen3-4B and 26.3% with Qwen3-8B. This substantially exceeds the strongest baselines at each scale, which achieve only 14.8% (PPO) and 19.8% (GRPO), respectively. This performance gap is sustained in the repository-level SWE-bench Verified environment. For Qwen3-8B, LVRL reaches 20.2%, substantially exceeding GRPO (12.0%) and PPO (9.0%). Furthermore, scaling the policy to Qwen3-32B increases LVRL's performance to 30.9%, outperforming PPO's 24.9%. This demonstrates that LVRL remains highly effective even for large-capacity models. Beyond final performance, LVRL significantly improves overall learning efficiency. As illustrated in Figure 9, LVRL establishes an early lead in training reward and converges much faster across these environments. In contrast, GRPO struggles with slower convergence, and PPO requires substantially more environment interactions to learn a useful critic. This dynamic confirms that our step-level value guidance enhances sample efficiency.

The advantage of step-level credit assignment is most pronounced in the BabyAI setting. Crucially, because the base models' success rates are near 0%, initial rollouts yield uniformly zero rewards. This creates a severe cold-start bottleneck: GRPO advantages vanish without reward variance, and PPO's learned critic collapses to predicting zero everywhere. Consequently, both baselines fail to extract meaningful learning signals, scoring near 0% on both in-distribution (ID) and out-of-distribution (OOD) layouts. For LVRL, when all trajectories in a group fail, both endpoint anchors are zero, so the unnormalized advantages sum to zero for every trajectory. Nevertheless, interior estimates can vary, producing positive and negative step weights. This allows LVRL to use the critic's inherent semantic capacity to recognize and reward intermediate progress even when a trajectory ultimately fails. By providing this step-level signal, LVRL sidesteps the cold-start problem, allowing Qwen3-8B to achieve 75.0% ID success and 65.5% OOD success.

To understand these gains qualitatively, we analyze the step-level credit assigned by LVRL across environments. Unlike GRPO, whose relative advantages vanish entirely if all rollouts in a group fail, the frozen LLM critic in LVRL can still isolate useful intermediate progress. For instance,

Table 1: LVRL evaluation performance across three environments. BabyAI results are reported as ID / OOD success rates. Uncertainties denote standard deviations. Bold indicates the best mean for each environment and policy size. The frozen LLM value critic is Claude Opus 4.6.

	AppWorld		BabyAI		SWE-bench Verified	
Method	Qwen3-4B	Qwen3-8B	Qwen3-4B	Qwen3-8B	Qwen3-8B	Qwen3-32B
Base	4.2 ± 0.4	10.9 ± 1.0	0.0 ± 0.0 / 0.0 ± 0.0	0.7 ± 0.0 / 0.0 ± 0.0	2.9 ± 0.2	20.2 ± 1.2
+GRPO	10.7 ± 2.8	19.8 ± 3.8	0.3 ± 0.4 / 0.0 ± 0.0	2.7 ± 2.9 / 0.0 ± 0.0	12.0 ± 1.2	22.3 ± 2.1
+PPO	14.8 ± 0.8	17.5 ± 1.6	0.1 ± 0.3 / 0.0 ± 0.0	0.3 ± 0.3 / 0.0 ± 0.0	9.0 ± 0.4	24.9 ± 0.5
+LVRL (ours)	$\mathbf{24.3} \pm 1.7$	$\mathbf{26.3} \pm 0.5$	$\mathbf{69.0} \pm 2.2$ / $\mathbf{52.2} \pm 1.6$	$\mathbf{75.0} \pm 2.0$ / $\mathbf{65.5} \pm 3.0$	$\mathbf{20.2} \pm 1.3$	$\mathbf{30.9} \pm 1.6$

it assigns positive value differences to constructive steps, such as recovering from API parameter errors in AppWorld, opening a door to reveal a target in BabyAI, or successfully reproducing a bug in SWE-bench Verified, even these trajectories ultimately fail. Conversely, it assigns negative differences to local errors within successful trajectories, such as invalid file edits or incorrect API calls. By explicitly rewarding intermediate semantic progress and penalizing localized mistakes, LVRL avoids GRPO's uniform trajectory-level credit while circumventing PPO's dependence on learning a critic from sparse returns. Detailed training trajectory examples illustrating these value changes are shown in Appendix G.

However, this dense, zero-shot supervision introduces a computational trade-off. As detailed in Appendix B.1, querying Claude Opus 4.6 for trajectory evaluation incurs an inference cost of approximately 9K to 20K input tokens and 0.6K to 1.9K output tokens per target trajectory, depending on the environment context length. Importantly, our framework is not tied to a single proprietary API. As shown in our ablations (Section 4.5), alternative strong models like GLM-5.2 can also yield competitive policy improvements. Consequently, scaling this method highlights a promising direction for future work: distilling this high-quality signal into a smaller, more efficient local critic.

4.3 Does LVOPD Improve On-Policy Distillation?

We evaluate whether the step-level advantages produced by our frozen LLM value critic improve on-policy distillation. As established in Section 4.1, the three evaluated methods form a progression in signal granularity. Pure OPD relies entirely on teacher supervision without environment reward. Trajectory-gated OPD adds a GRPO policy loss and uses its trajectory-level advantage to gate distillation. LVOPD finally resolves both the policy-gradient signal and the gate to the step level.

As shown in Table 2, LVOPD yields consistent improvements over standard distillation baselines on AppWorld. For Qwen3-4B, LVOPD achieves 20.2% success, outperforming both Pure OPD (17.0%) and Trajectory-gated OPD (19.2%). A similar trend is observed for Qwen3-8B, where LVOPD reaches 23.3%, compared to 22.2% for Pure OPD and 21.2% for Trajectory-gated OPD. These results indicate that step-level gating provides a more reliable distillation signal than uniform or trajectory-level approaches. This generalization advantage is closely aligned with the observed training dynamics. As illustrated in Figure 10, while pure OPD exhibits rapid initial learning, LVOPD surpasses it after approximately 100 training steps and achieves the highest final training reward across both model scales. This trajectory suggests that although uniform distillation provides dense early guidance, it eventually restricts the student's policy space. By selectively gating supervision, LVOPD avoids this plateau and continues to optimize based on valid student exploration.

Despite these gains, the efficacy of value-guided distillation remains strictly bounded by the underlying capability of the chosen LLM value critic. When evaluated on the BabyAI environment, all OPD variants, including LVOPD, completely fail to generalize, yielding success rates near 0% (Appendix E). The failure of pure OPD indicates that token-level supervision from Qwen3-235B model alone is insufficient for BabyAI environments. LVOPD cannot compensate through its value signal, because Qwen3-235B also serves as the frozen LLM critic, and a weaker critic yields noisier advantage signals, as our ablations show (Section 4.5). We hypothesize that swapping in a more capable LLM critic (e.g., Claude Opus 4.6) for value prediction while keeping the distillation teacher fixed could resolve this, which remains a direction for future work.

Table 2: OPD results on AppWorld. The frozen LLM critic is Qwen3-235B.

Method	Qwen3-4B	Qwen3-8B
Base	4.2 ± 0.4	10.9 ± 1.0
+OPD	17.0 ± 0.3	22.2 ± 0.7
+Trajectory-gated	19.2 ± 1.0	21.2 ± 1.0
+LVOPD (ours)	$\mathbf{20.2} \pm 1.8$	$\mathbf{23.3} \pm 1.6$

Table 3: In-context calibration ablation.

Method	Qwen3-8B
GRPO	19.8 ± 3.8
LVRL (ours)	$\mathbf{26.3} \pm 0.5$
w/o FS, anchor	22.1 ± 0.5

Table 4: Choice of frozen LLM critic.

LLM critic	Qwen3-4B
Claude Opus 4.6	$\mathbf{24.3} \pm 1.7$
Qwen3-235B	9.8 ± 2.3
GLM-5.2	22.3 ± 4.1

4.4 Validating Step-Level Value Signals

To validate the accuracy of the frozen LLM critic, we evaluate whether its predicted step-level value differences align with true changes in the probability of trajectory success. Because environments like AppWorld lack step-level, policy-dependent value annotations, we construct reference labels using extensive Monte-Carlo (MC) rollouts.

We use three checkpoints from a single AppWorld training run, taken at approximately 25%, 50%, and 75% of training to cover the early, middle, and late stages. For each checkpoint we randomly select 30 tasks and sample one successful and one failed trajectory per task under that checkpoint, giving 180 trajectories with $3{,}899$ states in total. For each recorded state s_t, we replay the original action prefix and sample $M = 32$ independent continuations from the same checkpoint. The fraction that succeed is our Monte-Carlo value estimate $V^{\mathrm{MC}}(s_t)$, and $\Delta_t^{\mathrm{MC}} = V^{\mathrm{MC}}(s_{t+1}) - V^{\mathrm{MC}}(s_t)$ measures the change across a transition. We label a transition a *pitfall* when $\Delta_t^{\mathrm{MC}} \leq -3\sigma$ and a *recovery* when $\Delta_t^{\mathrm{MC}} \geq 3\sigma$, where $\sigma = 0.125$ is the largest possible standard deviation of the difference between two independent estimates at $M = 32$ as detailed in Appendix F.2. Transitions with $|\Delta_t^{\mathrm{MC}}| < \sigma$ are *MC-neutral*: their estimated change is indistinguishable from sampling noise. This yields 76 pitfalls, 38 recoveries, and $1{,}533$ MC-neutral transitions.

For evaluation, let $\Delta_t^{\mathrm{pred}} = V_\phi(s_{t+1}) - V_\phi(s_t)$ denote the predicted value difference. The group-level normalization (Section 3.2) used during training divides this difference by a positive constant and therefore preserves its sign. Thus, we consider a step t to be a pitfall when $\Delta_t^{\mathrm{pred}} < 0$ and a recovery when $\Delta_t^{\mathrm{pred}} > 0$. Our analysis reveals that the predicted value change has the same sign as the Monte-Carlo change for 72% of pitfalls and 66% of recoveries. In comparison, it decreases on only 8% of MC-neutral transitions and increases on 22%. These substantial margins provide strong empirical evidence that the frozen LLM critic extracts meaningful, step-level outcome signals rather than just generating random noise.

We further compare LVRL against a model-free *error-marker* baseline for pitfall detection. This baseline labels a transition as a pitfall simply if the environment output contains text such as `Traceback`, `Execution failed`, or `No code available to execute`. The baseline detects only 13% of the MC pitfalls and flags 3% of the MC-neutral transitions. This demonstrates that explicit execution failures alone cannot fully account for the pitfall detection achieved by our predicted value signal. As illustrated in Appendix F.4.2, a step may produce an API error yet establish the correct task logic. The value model correctly identifies such decisions as positive progress, distinguishing the trajectory from other failed rollouts.

To understand where the remaining errors occur, we group the 114 Monte-Carlo events in ten bins by their position in the trajectory. As shown in Figure 11, agreement generally increases later in a trajectory. As a coarse summary, the sign-agreement rate increases from 44% in the first third to 78% in the middle third and 88% in the final third. In particular, the model assigns no value change to 29% of early events, compared with only 4% of late events, indicating that it often fails to react to events early in a trajectory. This pattern suggests that the predicted signal becomes more reliable as the trajectory progresses. We further analyze these early actions where the critic and the Monte-Carlo estimates diverged. Early disagreements frequently happen in useful preparatory actions, such as inspecting API documentation, logging in, or retrieving task data. In 12 out of 19 early disagreements, the LLM critic interprets the action as semantic progress and either maintains or increases its value prediction, whereas the empirical MC value actually decreases. This pattern reveals that the frozen LLM critic's value estimates are not strictly on-policy. While it correctly identifies useful semantic progress, it overestimates the current student's ability to successfully complete the task.

4.5 Ablation Studies

In-Context Calibration for the frozen LLM Value Critic. Our LLM critic uses information from the rollout group to calibrate its value estimates to the current policy. The group-average return $V(S_0)$ and final reward $R(\tau_i)$ anchor the initial-state value and final-state value, respectively, while k randomly sampled rollouts ($\{\tau_j, \tau_k, ...\}$) provide contemporary same-task examples. Because these components jointly serve to ground the critic's expectations in empirical outcomes, we ablate these context sources by providing only the current trajectory τ_i to the critic. As shown in Table 3, removing this calibration context reduces the AppWorld success rate from 26.3% to 22.1%.

While this ablated variant still outperforms the standard GRPO baseline (19.8%), the full method achieves the highest mean success rate, firmly validating the utility of in-context calibration. Although a fine-grained ablation isolating the endpoint anchors from the exact number of context examples is left to future work, we hypothesize that performance scales positively with context size. Specifically, incorporating more few-shot examples should yield a richer, more accurate representation of the current policy, thereby further enhancing the critic's value estimation.

Choice of the frozen LLM Value Critic. We evaluate how the underlying capability of the frozen LLM critic impacts downstream policy learning. Using LVRL, we train Qwen3-4B on AppWorld with different models as the frozen LLM value critic: Qwen3-235B, GLM-5.2 (Zeng et al., 2026), and Claude Opus 4.6. All other training settings remain fixed across three independent seeds per critic. As shown in Table 4, while all evaluated critics successfully improve upon the base model's 4.2% success rate, the magnitude of this improvement is heavily bottlenecked by the critic's inherent reasoning capacity. For instance, using Qwen3-235B as the critic yields a modest success rate of 9.8%. In contrast, substituting it with more capable models dramatically unlocks the student's learning potential, reaching 22.3% with GLM-5.2 and 24.3% with Claude Opus 4.6. This capability gap empirically validates our hypothesis from Section 4.3: the effectiveness of step-level credit assignment is fundamentally bottlenecked by the reasoning capacity of the frozen critic. Furthermore, the comparably high performance of GLM-5.2 and Claude Opus 4.6 demonstrates that LVRL generalizes well across different frontier model families.

5 Conclusion and Discussion

This work introduces LLM Value Function RL (LVRL), a method for dense credit assignment in LLM agent reinforcement learning. Instead of training a separate value model, LVRL uses a frozen LLM to estimate the value of intermediate states in a completed trajectory. It makes these estimates policy-aware by conditioning the LLM on rollout statistics from the current policy via in-context learning. By anchoring the initial and terminal values to the group success rate and the actual environment outcome, the consecutive value differences telescope to the group-relative return. This process safely redistributes the actual reward across individual steps, providing an advantage signal that distinguishes progress, mistakes, and recoveries without additional environment interactions. We also introduce LLM Value Function OPD (LVOPD), which reuses this step-level advantage to gate on-policy distillation, directing teacher supervision strictly to actions that decrease the estimated value. Across agent environments spanning tool use, software engineering, and spatial navigation, LVRL outperforms GRPO and PPO baselines, while LVOPD improves upon uniform and trajectory-gated distillation baselines.

Despite these strong results, our approach has several limitations. First, the method heavily relies on the capability of the frozen LLM critic to judge task progress. As shown in our experiments, a weaker estimator yields a noisier value signal and smaller downstream gains. For instance, our LVOPD experiments relied on Qwen3-235B as both the teacher and critic. Evaluating LVOPD with a stronger critic, such as Claude Opus 4.6, could potentially unlock much higher distillation performance, which we leave to future work. Second, LVRL introduces additional inference overhead. For every trajectory, querying a large LLM to predict values for each step during the RL update limits deployment when training throughput or compute budget is constrained. Future work can investigate reducing this cost, for example, by querying the estimator only when its value signal is highly uncertain or by distilling its step-level knowledge into a smaller, learned critic. Finally, we designed LVRL specifically for agentic environments where states are naturally expressed in text. Extending this step-level credit assignment to non-agentic RL domains, such as complex mathematical reasoning, remains an open and promising direction.

AI USE STATEMENT

We acknowledge the use of large language models during the preparation of this paper. Specifically, we used these tools to assist in writing and debugging experimental scripts, and to refine the grammar, phrasing, and overall readability of the manuscript. The authors carefully reviewed and edited all generated content and assume full responsibility for the integrity of the experimental results and the final publication.

ETHICS STATEMENT

This work focuses on improving credit assignment for autonomous LLM agents within simulated, sandboxed environments (AppWorld, BabyAI, and R2E-Gym). While advancing the capabilities of autonomous agents inherently carries broad dual-use risks regarding automated digital interactions and code execution, our study focuses on learning methods and utilizes public, open-weights models. We did not recruit human participants nor collect personal data for training and evaluation.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide comprehensive details of our experimental setup in the Appendix. Specifically, environment configurations and evaluation metrics are detailed in Appendix A; training hyper-parameters, and optimization budgets are provided in Appendix B; and all exact prompt templates for value estimation are listed in Appendix Appendix C.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Mohammad Sadegh Akhondzadeh, Vijay Lingam, Atula Tejaswi, Chanakya Ekbote, Sujay Sanghavi, and Aleksandar Bojchevski. Reward-gated on-policy distillation. *arXiv preprint arXiv:2607.04037*, 2026.

Anthropic. Introducing claude opus 4.6. `https://www.anthropic.com/news/claude-opus-4-6`, February 2026. Accessed September 19, 2026.

Chishui Chen, Yaoyou Fan, Te Sun, Yi Yang, Chenghao Sun, Delin Mao, Hongbo Qiao, Zuowei Zhang, Junxi Wang, Chenxing Sun, et al. Look ahead before you distill: Future trajectory validation of teacher guidance for agentic on-policy distillation. *arXiv preprint arXiv:2608.01953*, 2026.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=QXEhBMNrCW`.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=5h0qf7IBZZ`.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

Nan Jia, Haojin Yang, Xing Ma, Jiesong Lian, Shuailiang Zhang, Weipeng Zhang, Ke Zeng, Xunliang Cai, and Zequn Sun. Asymmetric on-policy distillation: Bridging exploitation and imitation at the token level. *arXiv preprint arXiv:2605.06387*, 2026.

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.

Zhewei Kang, Aosong Feng, Sergey Levine, Dawn Song, and Xuandong Zhao. Vimpo: Value-implicit policy optimization for llms. *arXiv preprint arXiv:2606.20008*, 2026.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29557–29590. PMLR, 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: Step-level advantage assignment for long-horizon agents via trajectory graph. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4709–4725, Rabat, Morocco, March 2026a. Association for Computational Linguistics. ISBN 979-8-89176-386-9. doi: 10.18653/v1/2026.findings-eacl.247. URL https://aclanthology.org/2026.findings-eacl.247/.

Xiaozhe Li, Tianyi Lyu, Yang Li, Yichuan Ma, Peiji Li, Linyang Li, Qipeng Guo, Dahua Lin, and Kai Chen. What and when to distill: Selective hindsight distillation for multi-turn agents. *arXiv preprint arXiv:2605.19447*, 2026b.

Yang Li, Zhichen Dong, Yuhan Sun, Weixun Wang, Shaopan Xiong, Yijia Luo, Jiashun Liu, Han Lu, Jiamang Wang, Wenbo Su, et al. Attention illuminates llm reasoning: The preplan-and-anchor rhythm enables fine-grained policy optimization. *arXiv preprint arXiv:2510.13554*, 2025.

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.

Wenze Lin, Jiale Zhao, Xitai Jiang, Songde Rao, Yining Li, Shenzhi Wang, Bingxiang He, and Gao Huang. On-policy distillation with verifiable reward. *arXiv preprint arXiv:2608.24696*, 2026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like training: A critical perspective. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.20783.

Chiyu Ma, Shuo Yang, Kexin Huang, Jinda Lu, Haoming Meng, Shangshang Wang, Bolin Ding, Soroush Vosoughi, Guoyin Wang, and Jingren Zhou. FIPO: Eliciting deep reasoning with future-KL influenced policy optimization. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=9fqVC3alVX.

Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, et al. Vision language models are in-context value learners. In *International Conference on Learning Representations*, volume 2025, pp. 33984–34009, 2025.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287. Morgan Kaufmann, 1999.

Changdae Oh, Wendi Li, Seongheon Park, Samuel Yeh, Tanwi Mallick, and Sharon Li. Neglected free lunch from post-training: Progress advantage for llm agents. *arXiv preprint arXiv:2606.26080*, 2026.

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

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

Zikang Shan, Han Zhong, Liwei Wang, and Li Zhao. Bringing value models back: Generative critics for value modeling in llm reinforcement learning. *arXiv preprint arXiv:2604.10701*, 2026.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. Hindsight credit assignment for long-horizon llm agents. *arXiv preprint arXiv:2603.08754*, 2026.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. Trace: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026.

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

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.

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*, 2025b.

Xian Wu, Kaijie Zhu, Ying Zhang, Lun Wang, and Wenbo Guo. rePIRL: Learn PRM with inverse RL for LLM reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=H9IIK4ySRD.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, et al. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195, 2026.

Haoran Xu, Hongyu Wang, Yifei Gao, Jiaze Li, Xiaofeng Zhang, and Xiaosong Yuan. Sg-opd: Sign-gated on-policy distillation via sign-consistency gating and phased teacher sampling. *arXiv preprint arXiv:2606.09304*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL https://openreview.net/forum?id=tvI4u1ylcqs.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Qiyong Zhong, Mao Zheng, Mingyang Song, Xin Lin, Jie Sun, Houcheng Jiang, Xiang Wang, and Junfeng Fang. Sod: Step-wise on-policy distillation for small language model agents. *arXiv preprint arXiv:2605.07725*, 2026.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhou24t.html.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*, 2025.

Yuhang Zhou, Lizhu Zhang, Yifan Wu, Mingyi Wang, Bo Peng, Jiayi Liu, Xiangjun Fan, and Zhuokai Zhao. Sage-opd: Selective agent-guided intervention for multi-turn on-policy distillation. *arXiv preprint arXiv:2606.19659*, 2026.

A Environment and Task Details

A.1 AppWorld

Environment. AppWorld (Trivedi et al., 2024) is a high-fidelity simulator of a personal digital environment in which the agent acts on behalf of a simulated user by writing Python against nine day-to-day applications (`amazon`, `file_system`, `gmail`, `phone`, `simple_note`, `splitwise`, `spotify`, `todoist`, `venmo`) exposed through documented APIs. Two meta-applications complete the interface: `api_docs`, which the agent queries to discover apps, list their APIs, and retrieve individual specifications, and `supervisor`, which holds the user's profile, account credentials, and the `complete_task` call that terminates an episode. The agent must discover the relevant APIs through `api_docs`, authenticate by retrieving credentials from `supervisor`, and only then compose the calls that satisfy the instruction.

Task splits. Policy optimization uses only the 90-task `train` split (30 scenarios $\times$ 3 variants). We report final performance on `test_challenge` (417 tasks), whose scenarios are disjoint from those in `train`.

Agent interface. Each episode is a multi-turn conversation over a stateful Python REPL. On every turn the policy emits free-form reasoning followed by a single fenced Python code block; the environment executes that block in a session that persists across turns, so variables, imports, and access tokens carry over. It returns either the captured standard output, an `Execution successful.` acknowledgement, or the Python traceback on error, with a 30-second timeout per block. An episode ends when the agent calls `apis.supervisor.complete_task()`, passing `answer=<value>` for question-answering tasks and no argument for state-changing ones, or when the turn budget is exhausted. Figure 2 shows an example.

Instruction prompt. We use the same prompt for both training and evaluation: a static preamble introducing the three `api_docs` discovery calls, a 14-turn demonstration dialogue on one Spotify task, and an eight-item instruction list (close every code block; reuse variables across turns; inspect an API specification before calling it; write one small code block per turn; page through paginated APIs via `page_index`; call `complete_task` when done).

Reward. Reward is sparse, binary, and awarded only at the end of an episode. When the agent calls `complete_task`, or when the turn budget runs out, AppWorld runs its own validation suite against the resulting database state, and we set $r = 1$ if and only if that suite passes in full. There is no intermediate shaping and no partial credit: the fraction of validation tests passed is logged for analysis but never enters the reward. Calling `complete_task` is neither sufficient nor free, since an episode that declares completion while the validator still fails scores zero, which is by far the most common failure mode we observe.

Evaluation configuration. At evaluation the trained policy is served with vLLM through an OpenAI-compatible endpoint, with the same prompt, action format, and binary success criterion as training. We sample at temperature 0.6 and take 3 rollouts per task, reporting pass@1 over all 417 `test_challenge` tasks. Two budgets are relaxed relative to training: the turn cap is 70, and the context cap is 40k tokens. Both budgets are held identical across all compared methods.

```
--- Task ---

Using these APIs, now generate code to solve the actual task:
My name is: Kyle Moore. My personal email is kyl_moor@gmail.com
and phone number is 6372953330.
Task: Name the artist most recommended to me on Spotify.

[agent] To find the artist most recommended to me on Spotify, I need to find
        the "recommendations" API. Let me check which APIs Spotify exposes.
        print(apis.api_docs.show_api_descriptions(app_name='spotify'))

[env]   Output: [... "show_recommendations : Get personalized song
        recommendations ..." ...]

[agent] print(apis.api_docs.show_api_doc(app_name='spotify',
                                         api_name='show_recommendations'))

[env]   Output: {"app_name": "spotify", "api_name": "show_recommendations",
        "parameters": [{"name": "access_token", "required": true, ...}]}

[agent] I need to page through all the songs and extract the artist names.
        page_index = 0; songs = []
        while page_index < 10:
            song_page = apis.spotify.show_recommendations(
                access_token=spotify_access_token, page_index=page_index)

[env]   Execution failed. Traceback:
          NameError: name 'spotify_access_token' is not defined

[agent] I need to get the access token first. Let's log in to Spotify.
        spotify_password = [p for p in passwords
                            if p["account_name"] == "spotify"][0]["password"]

[env]   Execution failed. Traceback:
          NameError: name 'passwords' is not defined

[agent] I need to get the passwords first using the `supervisor` app.
        passwords = apis.supervisor.show_account_passwords()

[env]   Execution successful.

                                [ ... ]

[agent] apis.supervisor.complete_task(answer=most_recommended_artist)

[env]   Execution successful.        -> validation suite passes, r = 1
```

Figure 2: An excerpt from a successful AppWorld episode (Qwen3-4B, task `287e338_1`), abridged and reflowed; the enclosing Python fences are omitted for space. The REPL is stateful, so the agent recovers from the two `NameErrors` by fetching credentials and re-establishing the access token in later turns rather than restarting. Reward is granted only after `complete_task` triggers AppWorld's validation suite.

Table 5: BabyAI training and evaluation pools. Optimal actions include rotations, forward movements, door toggles, and the final pickup. The three splits use disjoint seed ranges and share no layouts.

Split	Door distance	Layouts	Median optimal actions	Max optimal actions
Train	1	95	14	34
	2	95	24	48
	3	95	32	60
Evaluation ID	1	100	14	36
	2	100	23	46
	3	100	32	51
Evaluation OOD	4	100	40	62
	5	100	51	71

A.2 BabyAI Task Construction and Evaluation

Environment and task construction. We modify the BabyAI `Pickup` level to use a 19×13 grid of six 7×7 rooms in a 2×3 layout, where adjacent rooms may be connected by doors. We choose this layout to provide a challenging but learnable navigation task for the models in our study. All doors start closed. Each layout contains the same inventory of 18 objects: three types (ball, box, key) crossed with six colors, one of each. Each instruction has the form *"pick up the <color> <object>"* and uniquely identifies one target object. Figure 3 shows three layouts drawn from the evaluation pools at door distances one, three, and five. Task format and room structure are identical throughout; only the navigation horizon changes.

Difficulty control. We control the navigation horizon through *door distance*: the number of doors along the shortest valid path from the agent's start cell to a cell adjacent to the target, computed over the full grid with walls, objects, and door connectivity taken into account. Every task has door distance at least one, so the target is never in the agent's starting room. Table 5 shows that optimal trajectory length grows consistently with door distance. A fully connected 2×3 room graph has diameter three, but generated layouts do not connect every adjacent pair, so some tasks require detours and reach door distances of four and five; these form the out-of-distribution split.

Agent interface. The policy interacts with BabyAI entirely through text. At each turn, the agent receives a text description of its 7×7 egocentric view, the view covers the cells ahead of and beside the agent but nothing behind it, and walls block vision. Then the agent emits a short reasoning statement followed by exactly one primitive action in an `<action>` tag, chosen from `left`, `right`, `forward`, `pick`, `drop`, and `toggle`.

Figure 4 gives the protocol and an abridged successful episode.

Reward. Reward is sparse and binary: the agent receives $r = 1$ on the step where it picks up the target object and $r = 0$ on every other step. An episode terminates as soon as success is detected, or when the step budget is exhausted, whichever comes first. Picking up a *wrong* object neither terminates the episode nor yields reward: the instruction is verified against the object currently carried, so the agent must `drop` the wrong object before it can pick up another. A turn whose reply contains no parseable `<action>` tag is executed as a no-op. It still consumes one step of the budget and is recorded in the dialogue history with an explicit "no valid action found" message.

Evaluation configuration. We evaluate policies on two held-out sets: an in-distribution (ID) set and an out-of-distribution (OOD) set, defined relative to the door distances used during training. The training layouts have door distances in $\{1, 2, 3\}$. The ID set contains 300 layouts, with 100 at each of these three distances, and tests generalization to unseen spatial configurations within the training distance range. The training, ID, and OOD splits use disjoint seed ranges and share no layouts.

The OOD set contains 200 layouts, with 100 each at door distances 4 and 5. Neither distance appears in training, so solving these tasks requires navigating across more doors than in the training tasks. The grid dimensions, object inventory, instruction format, observation interface, and action space

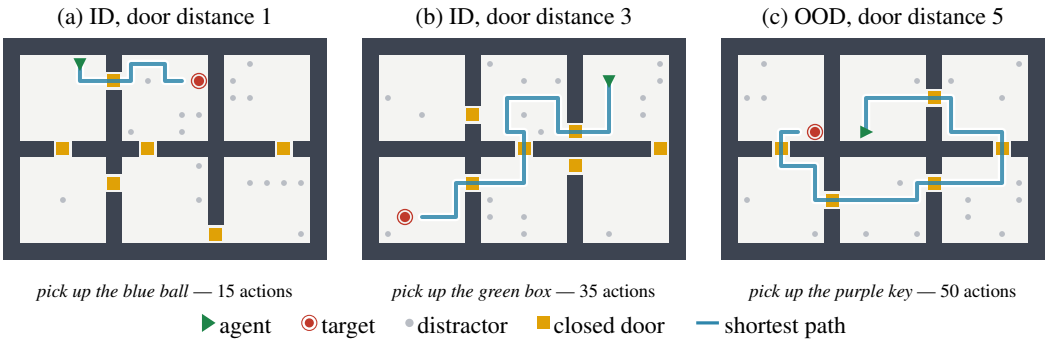


Figure 3: Representative BabyAI evaluation layouts at increasing door distance. The blue line is the shortest collision-free path from the agent to a cell adjacent to the target; the action count below each panel additionally includes turns, door toggles, and the final pickup. The path crosses one, three, and five doors respectively, which is what door distance measures. Every layout holds the same inventory of 18 objects, so distractor identities carry no information and only the spatial arrangement and the choice of target vary. The agent never observes this map: it receives a textual egocentric description (Figure 4).

are identical across all three splits. The OOD evaluation therefore tests generalization to longer navigation horizons under the same task definition and interaction rules.

All evaluated policies receive the same budget of 128 environment steps per episode, roughly $1.8\times$ the longest optimal trajectory in the OOD pool. This budget leaves additional steps for exploration and recovery beyond the optimal solution length. An episode terminates when the agent picks up the target object or exhausts the step budget. We report success rate as the fraction of evaluation episodes in which the agent picks up the target within this budget.

```
--- System protocol (abridged) ---

You are an agent in a BabyAI grid world. At each turn, you receive a textual
description of your egocentric view after "State:" and must select one action
to complete the task.

Available actions:
left / right: turn 90 degrees in place
forward:      move one cell forward
pick:         pick up the object directly ahead
drop:         drop the carried object
toggle:       open, close, or unlock the door directly ahead

Walls and objects block movement. To pick up an object, move to the adjacent
cell so that the object is directly ahead. If the target is not visible,
turn left or right to scan the room.

Respond with one or two short reasoning sentences, followed by exactly one
action in the following format: <action>forward</action>

--- Example episode (Qwen3-8B, 23 steps, success) ---

[env]  Task: pick up the red box
       State: You see a wall 2 steps forward. You see a wall 1 step left.

[agent] The red box is not in view, so I need to turn to scan the room.
        <action>left</action>

[env]  OK: executed `left`.
       State: You see a wall 1 step forward. You see a wall 2 steps right.
       You see a closed red door 3 steps left and 1 step forward.

[agent] The red door may lead to the box, so I should move toward it.
        <action>forward</action>

                              [ ... ]

[env]  OK: executed `right`.
       State: You see a wall 1 step left.
       You see a closed red door 1 step forward.

[agent] The red door is directly ahead, so I should open it.
        <action>toggle</action>

                              [ ... ]

[agent] The red box is directly ahead, so I should pick it up.
        <action>pick</action>

[env]  OK: executed `pick`.
       State: You carry a red box. You see a blue key 3 steps forward.
       You see an open red door 1 step right.
```

Figure 4: BabyAI interaction protocol and an excerpt from a successful episode. The policy sees only textual egocentric observations, states its reasoning briefly, and selects one primitive action per turn.

A.3 R2E-GYM AND SWE-BENCH VERIFIED

Environment. R2E-Gym (Jain et al., 2025) is a collection of executable software-engineering environments procedurally curated from real Python repositories. Each instance provides a natural-language issue report and a Docker image containing the repository snapshot at the corresponding commit, with the project and its dependencies pre-installed under `/testbed`; the agent's task is to edit the repository so that the issue is resolved. The scaffold exposes the sandbox through four tools:

- `search` (grep-style lookup over a file or directory),
- `file_editor` (view files with line numbers or a condensed skeleton, and `create`, `str_replace`, `insert`, `undo_edit` edits),
- `execute_bash` (run a shell command in the container),
- `finish` (declare the episode complete).

The agent must localize the fault from the issue text alone, edit the source, and typically write and run its own reproduction script to check the fix; each instance also ships with an execution-based test suite that is used only for reward and is never visible to the policy.

Task splits. Policy optimization uses executable instances from the R2E-Gym-Subset training pool, whose repositories are disjoint from those in SWE-bench Verified (Jain et al., 2025). We report final performance on SWE-Bench Verified (Jimenez et al., 2024), the 500-task human-validated subset of SWE-Bench, which shares the same instance format (issue plus containerized repository) and therefore plugs into the same scaffold unchanged; since its repositories never appear in training, it measures transfer to unseen codebases rather than memorization.

Agent interface. Each episode runs in a fresh container started from the instance's image. On every turn the policy emits free-form reasoning followed by exactly one tool call in Hermes format, a `<tool_call>` tag wrapping a JSON object with the tool name and arguments; the tool schemas are injected once into the system message through the model's chat template. The environment executes the call with a 90-second timeout and returns the tool output as the next user message. A turn without a parseable tool call executes nothing: the agent receives a fixed corrective message and the turn still counts against the budget. Only the first call in a turn is executed. Per-turn generation is capped at $4{,}096$ tokens, and the episode ends when the agent calls `finish` or after 50 turns. Figure 5 shows an example.

Instruction prompt. We use the R2E-Gym (Jain et al., 2025) function-calling prompt unchanged for both training and evaluation: a one-sentence system message framing the agent as a programming assistant for issue resolution, and an instance message that embeds the issue inside `<github_issue>` tags followed by a fixed list of workflow tips (explore the repository first; write a `reproduce_issue.py` script with a timeout and run it to confirm the error; edit only non-test files; rerun the script after the fix; consider edge cases; every response must contain a function call).

Reward. Reward is sparse, binary, and computed only at the end of an episode. When the agent calls `finish`, or when the turn budget is exhausted, the harness runs the instance's held-out fail-to-pass and pass-to-pass tests against the final repository state (with a 500-second timeout), and $r = 1$ if and only if the full suite passes.

Evaluation configuration. At evaluation the trained policy is served with vLLM, and our harness is the same as in training: the same prompt, tool schemas, Hermes action format, observation packing, and context truncation, with thinking mode disabled for the Qwen3 backbone. The configuration is 50 turns, $4{,}096$ tokens per turn, temperature 1.0, top-p 1.0. For Qwen3-32B, the context length is 64k, while for Qwen3-8B, we maintain 32k. We take 3 independent rollouts over all 500 Verified tasks and report the mean and standard deviation of the resolve rate; budgets and sampling are held identical across all compared methods.

```
--- Issue (abridged) ---

Sitemaps without items raise ValueError on callable lastmod.
When sitemap contains no items, but supports returning lastmod for an item,
it fails with a ValueError: ... max() arg is an empty sequence ...

[agent] To address the GitHub issue: 1. explore the repository to locate
        get_latest_lastmod, 2. reproduce the ValueError, 3. fix it, 4. rerun.
        <tool_call>{"name": "search", "arguments":
          {"search_term": "def get_latest_lastmod", "path": "/testbed"}}</tool_call>

[env]   Found 4 matches for "def get_latest_lastmod" in /testbed:
        ./django/contrib/sitemaps/__init__.py (2 matches) ...

[agent] Let's view the exact lines where this method is defined.
        <tool_call>{"name": "file_editor", "arguments": {"command": "view",
          "path": "...", "view_range": [160, 260]}}</tool_call>

[env]   ERROR: Invalid view_range [160, 260]: end must be >= start and
        <= 258, or -1 to view until end.

[agent] Apologies for the error. I will view from line 160 to the end.
        <tool_call>{"name": "file_editor", "arguments": {"command": "view",
          "path": "...", "view_range": [160, -1]}}</tool_call>

[env]   165  def get_latest_lastmod(self):
        ...
        170  return max([self.lastmod(item) for item in self.items()])
        171  except TypeError: ...

[agent] The max() call raises ValueError on an empty items() list. I will
        catch both TypeError and ValueError and return None.
        <tool_call>{"name": "file_editor", "arguments":
          {"command": "str_replace", "path": "...",
           "old_str": "        except TypeError:",
           "new_str": "        except (TypeError, ValueError):"}}</tool_call>

[env]   The file /testbed/django/contrib/sitemaps/__init__.py has been edited.

[agent] Now let's create a script to reproduce the scenario and verify.
        <tool_call>{"name": "file_editor", "arguments": {"command": "create",
          "path": "/testbed/reproduce_issue.py", "file_text": "..."}}</tool_call>

[env]   File created at /testbed/reproduce_issue.py. ...

[agent] <tool_call>{"name": "execute_bash", "arguments":
          {"cmd": "timeout 20 python /testbed/reproduce_issue.py"}}</tool_call>

[env]   Exit code: 0
        get_latest_lastmod returned: None
        No error occurred.

[agent] We have located the method, modified it, and confirmed the fix ...

[env]   No valid tool call found in your response. Every response MUST
        include exactly one function call formatted as <tool_call>...</tool_call>.

[agent] <tool_call>{"name": "finish", "arguments": {"command": "submit",
          "result": "Fixed by catching ValueError in get_latest_lastmod
           ..."}}</tool_call>

[env]   <<< Finished >>>          -> held-out test suite passes, r = 1
```

Figure 5: An excerpt from a successful SWE-Bench Verified episode (instance `django__django-16255`). The agent localizes the fault with `search`, recovers from an invalid `view_range` by re-reading the tool's error message, patches the file with `str_replace`, and validates the fix with its own reproduction script before submitting.

Table 6: Estimated cost per logged value-estimation query. Token counts are means computed with `cl100k_base`, an uncalibrated proxy for Claude tokenization. Each query scores one target trajectory. Input counts include peer summaries, but the cost of generating those summaries is excluded. Dollar estimates use standard Claude Opus 4.6 rates.

Environment / student	Queries	Input tokens	Output tokens	USD / query
AppWorld / Qwen3-4B	92,718	9.01k	0.66k	0.061
AppWorld / Qwen3-8B	99,968	9.43k	0.61k	0.062
BabyAI / Qwen3-8B	8,838	11.86k	1.90k	0.107
R2E-Gym / Qwen3-8B	8,030	20.35k	1.55k	0.140

B Training Details

B.1 Cost of value estimation

We estimate token usage from the full logged prompts and responses, using the `cl100k_base` tokenizer as a proxy because API token usage was not retained. Each query jointly estimates all state values for one target trajectory. At Claude Opus 4.6's standard rates of \$5 and \$25 per million input and output tokens, respectively, the estimated cost per trajectory is \$0.061–\$0.062 for AppWorld, \$0.107 for BabyAI, and \$0.140 for R2E-Gym. These estimates exclude separate trajectory-summarization calls and do not apply caching or batch discounts.

We use the same model for rollout summarization and value prediction. For example, in LVRL, Claude Opus 4.6 is used in both cases. The summarization cost per trajectory is similar to value estimation as their number of input and output tokens are similar.

B.2 LVRL

Student policies. We initialize the student policies from the released Qwen3-4B, Qwen3-8B, and Qwen3-32B checkpoints and fine-tune all model parameters. The policies operate in non-thinking mode during both training and evaluation. AppWorld and BabyAI experiments use Qwen3-4B and Qwen3-8B, whereas R2E-Gym experiments use Qwen3-8B and Qwen3-32B. For each environment–model pair, GRPO, PPO, and LVRL use the same initial checkpoint, task-sampling procedure, rollout budget, number of optimization updates, and generation configuration.

Rollout and optimization budget. For AppWorld and BabyAI, training uses 64 parallel environments across eight GPUs, with eight environments per GPU. At each update, the 64 trajectories are organized into eight rollout groups of size $G = 8$. The trajectories within each group correspond to the same task and initial state. AppWorld, BabyAI and R2E-Gym are trained for 500, 150, and 100 updates, respectively. The same budgets are used for GRPO, PPO, and LVRL.

PPO critic. Our PPO baseline adds a `DeepCriticHead` to the student LM. The critic is a four-layer MLP with hidden width $2{,}048$, implemented in bfloat16 and attached to the LM hidden states. The policy and critic share the entire LM backbone. They are trained jointly using the same optimizer and backward pass, and gradients from the value loss update the shared backbone. We use $\gamma = 1.0$ and $\lambda = 0.95$ for GAE. Advantages are normalized before policy optimization. The value loss is a clipped mean-squared error between the predicted values and return targets, with coefficient 0.5. The critic architecture and hyperparameters are held fixed across the three tasks. GRPO uses the same rollout groups and policy-optimization configuration but does not train a critic.

Checkpoint selection and evaluation. For AppWorld, we select the final checkpoint for each training run and evaluate it on the Test Challenge split. For BabyAI and R2E-Gym, we use the final saved checkpoint, defined as the checkpoint with the largest training-update index. Each selected checkpoint is evaluated in three independent stochastic evaluation runs. For AppWorld and BabyAI, each reported seed result is the average of its three evaluation runs, and we report the mean and standard deviation across three independently trained seeds. For R2E-Gym, we train one seed and report the mean and standard deviation across its three evaluation runs.

Table 7: Configured rollout and optimization settings for LVOPD. The effective optimization batch size is the product of the per-rank minibatch size, gradient-accumulation steps, and training world size.

Setting	AppWorld	BabyAI
Training updates	200	150
Rollout-step budget	50	128
Training nodes	1	2
Training world size	8	16
Minibatch size per rank	4	4
Gradient-accumulation steps	8	4
Effective optimization batch size	256	256
Instruction-token limit	16,000	16,000
vLLM maximum sequence length	40,000	40,000

B.3 LVOPD

Student policies and frozen teacher. We study LVOPD on AppWorld and BabyAI with Qwen3-4B and Qwen3-8B student policies, updating all student parameters. Student rollouts use non-thinking mode. Qwen3-235B-A22B-Thinking-2507-FP8 serves as both the frozen value estimator and the frozen distillation teacher, as described above. Neither role receives gradient updates.

Value-guided distillation. We use the gated objective defined in Section 3.3, with distillation coefficient $\beta = 1.0$. The value estimator produces continuous state-value predictions; consecutive differences are normalized by their within-group standard deviation to obtain $A_{i,t}$. The distillation weight is $g(A_{i,t}) = \max(-A_{i,t}, 0)$, shared by all tokens of an environment action. Because the teacher server returns log-probabilities only for its top-k_E tokens, with $k_E = 32$, we approximate the reverse KL on the student's support, which is natural since the reverse KL is an expectation under the student. At each prefix h, let $\mathcal{S}(h)$ be the student's top-k_S tokens, with $k_S = 16$, and let $\mathcal{T}(h)$ be the teacher's top-k_E tokens. For $v \in \mathcal{S}(h) \cap \mathcal{T}(h)$, we use the returned teacher probability $\pi_E(v \mid h)$. For $v \in \mathcal{S}(h) \setminus \mathcal{T}(h)$, we spread the teacher's residual mass uniformly over the tokens outside $\mathcal{T}(h)$, setting $\tilde{\pi}_E(v \mid h) = \big(1 - \sum_{u \in \mathcal{T}(h)} \pi_E(u \mid h)\big)/\big(|\mathcal{V}| - k_E\big)$, where $\mathcal{V}$ is the vocabulary. We renormalize both distributions over $\mathcal{S}(h)$ to obtain $\bar{\pi}_\theta$ and $\bar{\pi}_E$, and compute $\widehat{\mathrm{KL}}(h) = \sum_{v \in \mathcal{S}(h)} \bar{\pi}_\theta(v \mid h) \log\big(\bar{\pi}_\theta(v \mid h)/\bar{\pi}_E(v \mid h)\big)$, with gradients flowing only through the student. Setting $k_E > k_S$ ensures that most tokens in the student's support receive an exact teacher probability. Few-shot prompt compression is enabled for value estimation.

Rollout and optimization budgets. We configure eight environments per training rank and a rollout group size of $G = 8$. Student actions are sampled with temperature 1.0 and top-$p = 1.0$, without top-k sampling truncation. The training budgets are 200 updates for AppWorld and 150 for BabyAI; the environment-specific settings are summarized in Table 7. The minibatch and gradient-accumulation settings yield a nominal effective optimization batch size of 256 in both environments.

Optimization and implementation. We use a constant learning rate of 10^{-5} and one optimization epoch per collected rollout batch. The policy-ratio clipping parameter is 0.2, the TIS cap is 2.0, and the maximum gradient norm is 1.0. The entropy coefficient is zero. Training uses fully sharded data parallelism (`FULL_SHARD`) and gradient checkpointing. Student rollouts are generated by a separate synchronous vLLM sampler using processed log-probabilities. The frozen teacher is served separately with tensor parallelism of degree four and eight asynchronous teacher workers.

Training seeds and checkpoint selection. For each environment–model configuration, we train three independent seeds. For both AppWorld and BabyAI, we evaluate the final saved checkpoint from each run, defined as the checkpoint with the largest training-update index.

C PROMPTS FOR ROLLOUT SUMMARIZATION AND VALUE ESTIMATION

We provide the prompt templates for rollout summarization and continuous value prediction. Braced fields are filled with the corresponding task and trajectory data. The wording is preserved from the implementation; only line breaks are adjusted for readability.

Rollout summarization. The summarization prompt (Figure 6) converts a completed rollout into one short entry per step, retaining the action, its key result, and important state changes. Summaries are cached within the rollout batch and reused when the rollout serves as context for other target trajectories. The target trajectory itself is not replaced by a summary.

Value prediction. The value prompt (Figure 7) includes the task, the target trajectory's final outcome, its sequence of actions and environment responses, and context from other rollouts. A single query returns values and brief explanations for steps $0, \ldots, T$, where step 0 precedes the first action and step t follows action t. Initial and terminal values are anchored after prediction, as described in the method. The shared prompt retains the implementation's reference to APIs and a Python REPL.

Context and trajectory formatting. The {fewshot_section} field is constructed as shown in Figure 8. It reports the success rate of the full rollout group and includes summaries of selected peer rollouts, excluding the target. The peer block is repeated for each selected rollout; {n_same_outcome} counts group members with the target's outcome. The {formatted_steps} field lists each action followed by its environment response. All steps are retained, while long environment responses are truncated before insertion. The implementation uses a default cap of $4{,}000$ characters per response and appends a truncation marker.

```
You are condensing ONE rollout of an AI agent solving a task, so it can be used as a
compact in-context example for another model. Rewrite the trajectory into a short
numbered list | ONE line per step | keeping ONLY what matters for judging progress
toward the goal.

For each step keep: what the agent attempted (which app/API or action) and the KEY
result | whether it succeeded or failed, any error type (e.g. 401 unauthorized,
no-such-API, wrong parameter), and any important state change (e.g. obtained an access
token, found the password, located the note, retrieved N items, completed the task).
DROP boilerplate, long JSON dumps, API-doc listings, and verbose reasoning.

Task:
{task_description}

Final outcome: the agent {outcome} (final reward = {final_reward}).

Full trajectory ({total_steps} steps):
{formatted_steps}

Output ONLY the condensed steps, one per line, nothing else:
Step 1: <one or two sentences: action + key result>
Step 2: <...>
```

Figure 6: Rollout-summarization prompt used by the per_step compression mode. It receives a completed rollout and returns a concise action–result description for each step.

D TRAINING CURVES

Figure 9 and Figure 10 demonstrate the LVRL and LVOPD training curves, respectively. LVRL and LVOPD consistently outperform other baselines with respect to training efficiency.

E LVOPD BABYAI RESULTS

Table 8 shows the OPD results of BabyAI. All three OPD methods fail to generalize.

```
You are an expert at analyzing AI agent trajectories. An AI agent is attempting to solve a
task. The agent **{outcome}** this task (final reward = {final_reward}).

## Task
{task_description}
{fewshot_section}
## Full Trajectory to Analyze (Steps 1-{total_steps})
{formatted_steps}

## Instructions
For each point in the trajectory (step 0 = before any action, step 1 = after the first
action, ..., step {total_steps} = after the last action), estimate the probability
(0.0 to 1.0) that the agent will successfully complete the task from that point
onward. Consider:
- Whether the agent's approach is correct up to that step
- Whether any errors have been made that are hard to recover from
- How much progress has been made toward the goal
- How many steps remain

Respond with ONLY a JSON object in this exact format (no other text):
{
  "values": [
    {"step": 0, "value": <float>, "reasoning": "<brief explanation>"},
    {"step": 1, "value": <float>, "reasoning": "<brief explanation>"},
    ...
    {"step": {total_steps}, "value": <float>, "reasoning": "<brief explanation>"}
  ]
}
```

Figure 7: Continuous value-prediction prompt. The model sees the completed target trajectory and returns all state values in one response. The {fewshot_section} field is expanded in Figure 8. The JSON braces are shown as sent to the model, rather than doubled as in Python format strings.

```
## Current Policy's Performance on This Task
Group success rate: {n_success}/{group_size} rollouts succeeded. The trajectory below
is one of the {n_same_outcome} that {outcome}.

## Other Rollouts by the Current Policy on This Task
These are rollouts from the SAME task by the SAME current policy. Use them to
calibrate what 'good' and 'bad' look like FOR THIS POLICY. Identify where the target
trajectory diverges from the successful ones, and which steps match the failed ones.

### Rollout {j} | {SUCCESS_or_FAIL} (reward={reward_j}, {steps_j} steps)
{summary_j}
```

Figure 8: Template of the context inserted into {fewshot_section}. The final rollout block is repeated for each selected peer. This block is assembled by code and is part of the value-prediction request; it is not a separate LLM query.

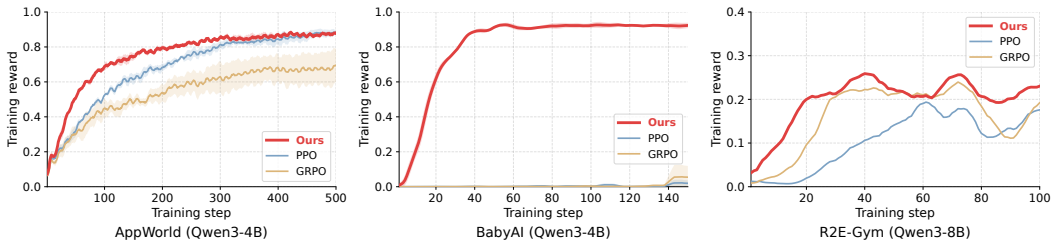


Figure 9: RL training reward curves across three training environments. For AppWorld and BabyAI, curves show the mean over three seeds; shaded regions denote one standard deviation.

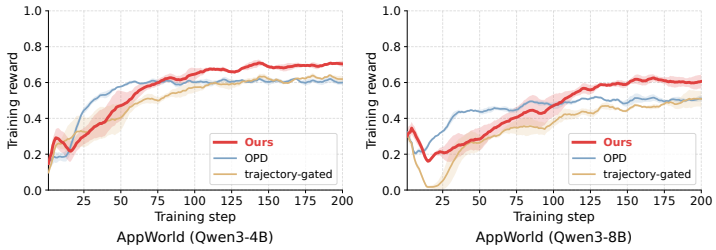


Figure 10: OPD training reward curves on AppWorld across model sizes. Curves show the mean over three seeds; shaded regions denote one standard deviation. Both gated variants consistently outperform the OPD baseline after roughly 100 training steps.

F Additional Analysis of Step-Level Value Predictions

F.1 Monte-Carlo Labeling Protocol

Value estimator. For a recorded state s_t we sample $M = 32$ independent continuations from the checkpoint policy and record whether each one completes the task. The estimator is the success fraction

$$V^{\mathrm{MC}}(s_t) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\left[\text{continuation } m \text{ succeeds}\right], \qquad \Delta_t^{\mathrm{MC}} = V^{\mathrm{MC}}(s_{t+1}) - V^{\mathrm{MC}}(s_t). \tag{3}$$

State reconstruction. AppWorld is stateful: the environment holds both a relational database and the live REPL namespace, and neither can be restored by prompting alone. We therefore reconstruct s_t by resetting the environment to its initial state for the task and re-executing every code cell in the recorded prefix $a_{0:t-1}$ in order, which restores the database side effects and the Python variable bindings the agent had built up. Continuations are then sampled from the reconstructed state, so all M rollouts branch from an identical environment. We verified reconstruction by checking that the replayed prefix reproduces the recorded environment responses; transitions where replay diverged were dropped before labeling.

F.2 Choice of the Event Threshold

Each V^{MC} is a binomial proportion over $M = 32$ samples, so its standard deviation is at most $\sqrt{0.25/M}$, attained at $V^{\mathrm{MC}} = 0.5$. Since $V^{\mathrm{MC}}(s_t)$ and $V^{\mathrm{MC}}(s_{t+1})$ are estimated from independent continuation sets, the standard deviation of their difference is bounded by

$$\sigma = \sqrt{2 \cdot \frac{0.25}{M}} = \sqrt{\frac{0.5}{32}} = 0.125. \tag{4}$$

We denote this maximum standard deviation by σ and label a transition as an event only when $|\Delta_t^{\mathrm{MC}}| \geq 3\sigma$. Table 9 reports the exact two-sided tail probability under the worst-case null distribution at $p = 0.5$. It also reports the estimated number of spurious events over the full evaluation

Table 8: Evaluation performance under on-policy distillation.

	BabyAI	
Method	Qwen3-4B	Qwen3-8B
Base	0.0 ± 0.0 / 0.0 ± 0.0	$\mathbf{0.7} \pm 0.0$ / 0.0 ± 0.0
+OPD	0.5 ± 0.1 / 0.0 ± 0.0	0.6 ± 0.3 / 0.0 ± 0.0
+Trajectory-gated	0.4 ± 0.2 / 0.0 ± 0.0	0.3 ± 0.3 / 0.0 ± 0.0
+LVOPD (ours)	$\mathbf{0.7} \pm 0.2$ / 0.0 ± 0.0	0.4 ± 0.1 / 0.0 ± 0.0

Table 9: Sampling noise for differences between two independent Monte-Carlo estimates with $M = 32$. The final column estimates the number of spurious events over the full evaluation set under the null.

Threshold	Minimum difference	Worst-case tail probability	Expected spurious events
1σ	$4/32$	0.3820	664
2σ	$8/32$	0.0599	72
3σ	$12/32$	0.0037	3.4

set, obtained by mixing the null distribution over the empirical state-value distribution. At 3σ, the expected number of noise-induced events is small relative to the 114 events observed. We therefore use 3σ to obtain a conservative event set.

F.3 Breakdown by Trajectory Position

We analyze the 114 high-confidence Monte-Carlo events by dividing each trajectory into thirds according to normalized step position. For each event, the predicted change is categorized as having the same sign as the Monte-Carlo change, being exactly zero, or having the opposite sign. Because the terminal value is anchored to the observed reward, a final transition could agree with the Monte-Carlo change by construction. However, we find that none of the 114 selected events is a final transition, so this anchoring does not affect the reported agreement: excluding final transitions leaves sign agreement unchanged at 70.2% (80/114) overall and 88.5% (23/26) in the final third.

Agreement increases across the three trajectory segments, while both zero and opposite-sign predictions become less frequent. The largest change is in zero predictions, which fall from 29% in the first third to 4% in the final third. The median predicted magnitude also increases fourfold, from 0.025 to 0.100, whereas the median Monte-Carlo magnitude increases more moderately. These results suggest that the value model is more conservative early in a trajectory and becomes more responsive as the trajectory progresses.

We also examine several task-level properties. The opposite-sign rate is 12% on Easy tasks ($n = 25$), 16% on Medium tasks ($n = 70$), and 11% on Hard tasks ($n = 19$). It is similarly stable when tasks are divided by reference-solution length: 13% for solutions with at most 40 lines and 16% for longer solutions. Other stratifications, including the number of apps and whether the task requires a returned answer, produce smaller differences over limited subgroups. We therefore do not find evidence that standard measures of task difficulty explain the position-dependent pattern.

Trajectory position is only a coarse proxy for how far an action is from its observable consequences. The results are consistent with long-range attribution being more difficult, but do not establish it as the cause. The qualitative examples below illustrate both the local evidence the value model uses successfully and the downstream effects it can miss.

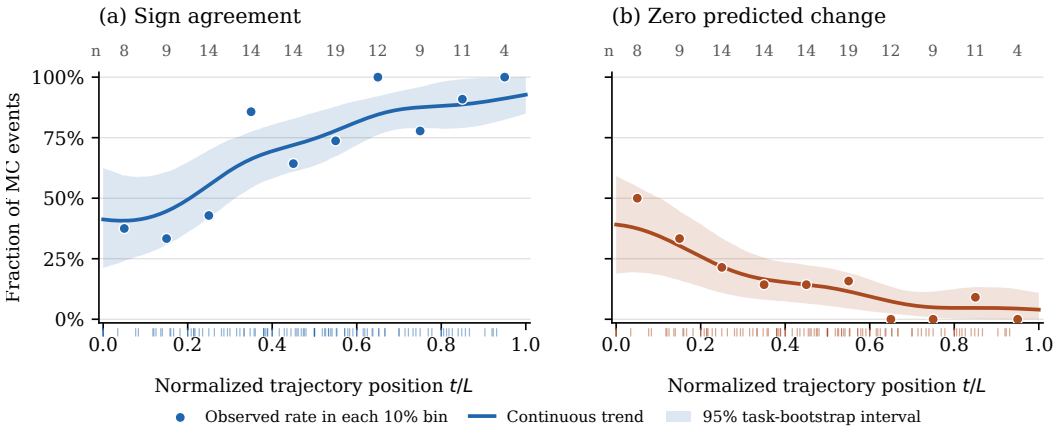


Figure 11: Agreement between predicted value changes and Monte-Carlo events across normalized trajectory position t/L, where L is the number of actions. (a) Fraction of MC events in each position bin for which the predicted value changes in the same direction as the MC estimate: a decrease for pitfalls and an increase for recoveries. (b) Fraction of MC events for which the model predicts no value change ($\Delta_t^{\text{pred}} = 0$), despite an MC increase or decrease large enough to meet the event threshold ($|\Delta_t^{\text{MC}}| \geq 3\sigma$).

Table 10: Step-level agreement by normalized trajectory position. *No change* denotes $\Delta^{\text{pred}} = 0$. Percentages may not sum to 100% because of rounding.

Position	Same sign	No change	Opposite sign
First third ($n = 34$)	44%	29%	26%
Middle third ($n = 54$)	78%	13%	9%
Final third ($n = 26$)	88%	4%	8%

F.4 Qualitative Examples

We present three representative transitions drawn from different tasks and two training checkpoints. Monte-Carlo values are estimated from 32 rollouts per state, and all reasoning excerpts are quoted verbatim. Table 11 summarizes the three transitions; each is then shown in full below.

Table 11: The three transitions examined in this section. The predicted value tracks the Monte-Carlo value on the first two transitions and moves in the wrong direction on the third.

	Task	Outcome	Δ^{MC}	Δ^{pred}	Detected
Pitfall, no execution error	`3c13f5a_2`	✗	-0.938	-0.70	✓
Recovery despite an error	`34d9492_2`	✓	$+0.438$	$+0.30$	✓
Missed pitfall	`27e1026_1`	✗	-0.500	$+0.05$	✗

F.4.1 Pitfall despite successful execution

Pitfall despite successful execution `3c13f5a_2 · traj. 638 · fails`

TASK

Divide the internet bill among the user's roommates and create Venmo requests with the exact description `internet bill for the last month.`

Step	Agent action → environment response	V^{MC}	V^{pred}
27	Creates three payment requests → *all three created successfully*	$0.94 \downarrow 0.00$	$0.80 \downarrow 0.10$

ACTION

```
description="internet bill for the last month"   # trailing period missing
```

VALUE MODEL

> *Payment requests were created but the description is 'internet bill for the last month' WITHOUT a period at the end. The task requires 'internet bill for the last month.' (with period). All successful rollouts used the exact description with the period. This mismatch likely causes failure.*

WHY IT MATTERS

The environment reports no execution failure, yet the action violates an exact-match requirement, and the requests cannot be un-created. Both values identify the transition as a severe pitfall, showing that the predicted value can catch a semantic error behind a successful execution.

F.4.2 Recovery despite an execution error

Recovery despite an execution error `34d9492_2 · traj. 207 · succeeds`

TASK

Organize vacation photographs into Rome, Santorini, and Berlin directories according to their creation dates.

Step	Agent action → environment response	V^{MC}	V^{pred}
13	Writes the date-to-destination mapping → *error: parameter is* `directory_path`	0.31↑0.75	0.40↑0.70

ACTION

```
vacation_directories = {"Rome": ..., "Santorini": ..., "Berlin": ...}
for directory in vacation_directories.values():
    if not apis.file_system.directory_exists(directory=directory, ...):
        apis.file_system.create_directory(directory=directory, ...)
```

VALUE MODEL

> *Agent wrote correct classification logic: March 2023→Rome, April 2023→Santorini, others (2022)→Berlin. Used correct directory names 'Rome', 'Santorini', 'Berlin'. This is the critical step where most other rollouts failed, and this agent got it right. However, execution failed due to wrong API parameter name.*

WHY IT MATTERS

The call fails, but the step fixes the classification rule that most rollouts get wrong, and the parameter name is trivial to repair. An execution-error rule would mislabel this transition: an error message does not imply that the state became less likely to succeed.

F.4.3 Missed pitfall with a downstream consequence

Missed pitfall with a downstream consequence `27e1026_1 · traj. 517 · fails`

TASK

Find the oldest released song across the user's Spotify song, album, and playlist libraries.

Step	Agent action → environment response	V^{MC}	V^{pred}
14	Concatenates records from the three Spotify sources → *no error*	0.84↓0.34	0.65↑0.70
15	Sorts the combined list by release date → `KeyError: 'release_date'`	0.34↓0.03	0.70→0.70

ACTION

```
all_songs = song_details + playlist_songs + album_songs
# playlist/album records carry 'release_date'; song-library records carry 'added_at'
```

VALUE MODEL

> *Got release dates for all playlist songs via show_song API. Good progress. [...] Got release dates for all album songs via show_song API. Now has all data needed.*

WHY IT MATTERS

The model verifies that each retrieval call succeeded, but never checks whether the three outputs share a schema, so it reads local data collection as progress. This is the characteristic failure mode: the immediate result of an action is verified while its support for later computation is not. It is consistent with the lower agreement observed earlier in trajectories, where the consequences of an action are less explicit at the time it is taken.

G EXAMPLES OF TRAINING TRAJECTORIES

We present three illustrative training cases in which the value model responds to consequential decisions. These selected examples complement the preceding Monte-Carlo analysis; they have no Monte-Carlo labels and are not included in its event counts. V_j^{pred} is the logged hindsight value after action j, and $\Delta_j^{\text{pred}} = V_j^{\text{pred}} - V_{j-1}^{\text{pred}}$. Tables show interior, unnormalized differences. Actions and observations are abridged and reflowed; omissions are marked or described. Value-model quotations are verbatim excerpts.

G.1 APPWORLD: A SEMANTIC ERROR DESPITE SUCCESSFUL EXECUTION

Deleting alarms when the user requested disabling them

TASK AND OUTCOME

I am going on a vacation. Move my go-to-sleep phone alarm to 1 hour later and disable the rest. Qwen3-4B; task `302c169_1`; nine actions; final reward $R = 0$.

PREDICTED VALUE TRACE

Step	Action and observed result	$V_{j-1}^{\text{pred}} \to V_j^{\text{pred}}$	Δ_j^{pred}
6	Retrieves all seven alarms	$0.90 \to 0.90$	0.00
7	Deletes six alarms instead of disabling them	$0.90 \to 0.00$	$\mathbf{-0.90}$
8	Moves the remaining alarm from 21:30 to 22:30	$0.00 \to 0.00$	0.00

ACTION AND OBSERVATION EXCERPTS

The login and alarm-retrieval prefix is omitted, as are the alarm-selection and print statements at step 8. All six deletion acknowledgements are retained; the final completion call is shown as outcome context.

```
[step 7: agent reasoning excerpt]
First, let's disable all alarms except the "Go to sleep" alarm.

[step 7: code]
for alarm in alarms:
    if alarm["label"] != "Go to sleep":
        disable_result = apis.phone.delete_alarm(
            access_token=login_result["access_token"],
            alarm_id=alarm["alarm_id"]
        )
        print(disable_result)
[env] {"message": "Alarm deleted successfully."}
      {"message": "Alarm deleted successfully."}
      {"message": "Alarm deleted successfully."}
      {"message": "Alarm deleted successfully."}
      {"message": "Alarm deleted successfully."}
      {"message": "Alarm deleted successfully."}

[step 8: code excerpt]
update_result = apis.phone.update_alarm(
    access_token=login_result["access_token"],
    alarm_id=go_to_sleep_alarm["alarm_id"],
    time="22:30"
)
[env] {"message": "Alarm updated successfully."}

[step 9]
apis.supervisor.complete_task()
[env] Execution successful.
```

VALUE MODEL (AFTER STEP 7)

> *Critical error: The agent DELETED all other alarms instead of DISABLING them. The task says 'disable the rest' but the agent used delete_alarm instead of update_alarm with enabled=False.*

INTERPRETATION

The calls execute successfully but perform the wrong state change: six alarm records are deleted instead of retained in a disabled state. Updating the sleep alarm correctly does not restore those records, and no restoration occurs before completion. The predictor places the value drop at the semantic error, which an execution-error indicator would miss.

G.2 BABYAI: RESOLVING AN INVENTORY BOTTLENECK

The same drop action fails or succeeds depending on the state

TASK AND OUTCOME

Pick up the blue ball. Qwen3-8B; 73 actions; final reward $R = 1$. The agent is carrying a distracting red key and can carry only one object.

PREDICTED VALUE TRACE

Step	Action and observed result	$V_{j-1}^{\text{pred}} \to V_j^{\text{pred}}$	Δ_j^{pred}
14	Tries to pick up the target while carrying a key	$0.38 \to 0.30$	-0.08
15	Drops toward the occupied target cell; no change	$0.30 \to 0.28$	-0.02
69	Turns toward an empty cell beside the target	$0.09 \to 0.12$	$+0.03$
70	Drops the key successfully; inventory becomes free	$0.12 \to 0.90$	$+\mathbf{0.78}$
71	Turns back to face the target	$0.90 \to 0.92$	$+0.02$

ACTION AND OBSERVATION EXCERPTS

Steps 1–13, 16–68, and 72 are omitted. Step 72 is a blocked forward move. Observations retain the relevant original clauses; the successful pickup at step 73 is shown only as outcome context.

```
[before step 14]
State: You carry a red key, [...] You see a blue ball 1 step forward [...]
[step 14] <action>pick</action>
[env] State: You carry a red key, [...] You see a blue ball 1 step forward [...]
[step 15] <action>drop</action>
[env] State: You carry a red key, [...] You see a blue ball 1 step forward [...]

[step 69] <action>right</action>
[env] State: You carry a red key, You see a wall 3 steps forward,
      You see a blue ball 1 step left [...]
[step 70] <action>drop</action>
[env] State: You see a blue ball 1 step left, [...]
      You see a red key 1 step forward [...]
[step 71] <action>left</action>
[env] State: [...] You see a blue ball 1 step forward [...]

[step 73] <action>pick</action>
[env] State: You carry a blue ball [...]
```

VALUE MODEL (AFTER STEP 70)

> *DROP SUCCEEDED! Red key is now 1 step forward, agent is no longer carrying anything. Blue ball is 1 step left. Just need to turn and pick up.*

INTERPRETATION

The occupied target cell prevents dropping the key in front of the ball. After turning toward a free cell, the same action releases the key and makes the target pickup possible. The large positive change occurs at this enabling action, three actions before success; earlier ineffective attempts receive negative changes despite the eventual successful outcome.

G.3 R2E-GYM: DISTINGUISHING AN INCORRECT PATCH FROM ITS CORRECTION

Two accepted edits with different semantic consequences

TASK AND OUTCOME

Fix `Image.paste` so that a 50×50 source image can be pasted into a 100×100 destination without an explicit box. Qwen3-8B; R2E-Gym training instance; 15 actions; final reward $R = 1$. The issue description is abridged.

PREDICTED VALUE TRACE

Step	Action and observed result	$V_{j-1}^{\text{pred}} \to V_j^{\text{pred}}$	Δ_j^{pred}
10	Sets the paste region to zero area; edit accepted	$0.70 \to 0.35$	$\mathbf{-0.35}$
11	Reruns the reproduction; the size error remains	$0.35 \to 0.40$	$+0.05$
12	Inspects the implementation again	$0.40 \to 0.45$	$+0.05$
13	Uses source-image dimensions; edit accepted	$0.45 \to 0.80$	$\mathbf{+0.35}$
14	Reruns the same reproduction; it exits normally	$0.80 \to 0.95$	$+0.15$

ACTION AND OBSERVATION EXCERPTS

Steps 1–9 and the final submission at step 15 are omitted; step 12 is summarized in the table. Tool wrappers, surrounding output, and unchanged arguments are omitted. The same reproduction script is used at steps 11 and 14.

```
[step 10: file_editor, str_replace arguments]
old_str: "box = (0, 0) + self.size"
new_str: "box = (0, 0, 0, 0)"
[env] The file /testbed/PIL/Image.py has been edited. [...]

[step 11: execute_bash]
timeout 20s python /testbed/reproduce_issue.py
[env] [...] ValueError: images do not match [...]

[step 13: file_editor, str_replace arguments]
old_str: "box = (0, 0, 0, 0)"
new_str: "box = (0, 0, im.size[0], im.size[1])"
[env] The file /testbed/PIL/Image.py has been edited. [...]

[step 14: execute_bash]
timeout 20s python /testbed/reproduce_issue.py
[env] Exit code: 0
```

VALUE MODEL (AFTER STEP 10)

> *Made an incorrect fix: changed box to (0,0,0,0) which is a 4-tuple of zeros, meaning it would paste into a zero-size region. This won't work but the agent can still recover.*

INTERPRETATION

Both edits are accepted by the tool, but the first creates a zero-area region and the reproduction still fails. The second uses the source-image dimensions, after which the same reproduction exits normally. The trajectory ultimately succeeds, yet the predictor assigns opposite signs to the two edits. This illustrates discrimination between patch semantics rather than tool-call validity; it does not establish correctness on untested inputs.