Verifiable Process Rewards for Agentic Reasoning

Huining Yuan*
Tsinghua University
China
yuanhuining0@gmail.com

Zelai Xu*
Tsinghua University
China
zelai.eecs@gmail.com

Huaijie Wang
Tsinghua University
China
wang-hx23@mails.tsinghua.edu.cn

Xiangmin Yi
Tsinghua University
China
yxm25@mails.tsinghua.edu.cn

Jiaxuan Gao
Tsinghua University
China
samjia2000@gmail.com

Xiao-Ping Zhang
Tsinghua University
China
xpzhang@ieee.org

Yu Wang†
Tsinghua University
China
yu-wang@tsinghua.edu.cn

Chao Yu†
Tsinghua University
China
yuchao@sz.tsinghua.edu.cn

Yi Wu†
Tsinghua University
China
jxwuyi@gmail.com

ABSTRACT

Reinforcement learning from verifiable rewards (RLVR) has improved the reasoning abilities of large language models (LLMs), but most existing approaches rely on sparse outcome-level feedback. This sparsity creates a credit assignment challenge in long-horizon agentic reasoning: a trajectory may fail despite containing many correct intermediate decisions, or succeed despite containing flawed ones. In this work, we study a class of densely-verifiable agentic reasoning problems, where intermediate actions can be objectively checked by symbolic or algorithmic oracles. We propose Verifiable Process Rewards (VPR), a framework that converts such oracles into dense turn-level supervision for reinforcement learning, and instantiate it in three representative settings: search-based verification for dynamic deduction, constraint-based verification for logical reasoning, and posterior-based verification for probabilistic inference. We further provide a theoretical analysis showing that dense verifier-grounded rewards can improve long-horizon credit assignment by providing more localized learning signals, with the benefit depending on the reliability of the verifier. Empirically, VPR outperforms outcome-level reward and rollout-based process reward baselines across controlled environments, and more importantly, transfers to both general and agentic reasoning benchmarks, suggesting that verifiable process supervision can foster general reasoning skills applicable beyond the training environments. Our results indicate that VPR is a promising approach for enhancing LLM agents whenever reliable intermediate verification is available, while also highlighting its dependence on oracle quality and the open challenge of extending VPR to less structured, open-ended environments.

*Equal contribution.
†Corresponding authors.
Project Page Code Models

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

KEYWORDS

Large Language Models, Reinforcement Learning from Verifiable Rewards, Process Reward

ACM Reference Format:
Huining Yuan, Zelai Xu, Huaijie Wang, Xiangmin Yi, Jiaxuan Gao, Xiao-Ping Zhang, Yu Wang, Chao Yu, and Yi Wu. 2026. Verifiable Process Rewards for Agentic Reasoning. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 15 pages.

1 INTRODUCTION

Reinforcement learning from verifiable rewards (RLVR) has recently emerged as a powerful paradigm for improving the reasoning abilities of large language models (LLMs) [4, 6]. By replacing subjective human preferences with objective correctness signals, RLVR enables models to optimize against rewards that are difficult to hack and easy to verify, such as exact answers in mathematical reasoning or unit-test outcomes in coding. Recent breakthroughs in mathematical reasoning [21] demonstrate that outcome-level verifiable rewards can drive models to discover complex reasoning behaviors.

However, most existing RLVR methods rely primarily on *outcome-level rewards*: the model receives feedback only after or completing an entire trajectory. While outcome-level verification is effective for single-turn tasks, it becomes insufficient in long-horizon agentic reasoning. As LLM research shifts toward agentic tasks involving tool use, interaction, and multi-turn planning [7, 34], an LLM agent must make a sequence of decisions, such as selecting actions, updating beliefs, maintaining constraints, or planning several steps ahead. A trajectory may fail despite many correct intermediate decisions, or succeed despite flawed ones. This creates a fundamental credit assignment problem: sparse terminal feedback cannot reliably identify which intermediate actions should be reinforced.

Process supervision offers a natural way to address this challenge by providing feedback at intermediate steps. Existing Process Reward Models (PRMs) [15, 26], however, often rely on learned reward models, LLM-as-a-judge evaluations, or Monte Carlo rollouts. Learned or generative process rewards may be noisy, biased,

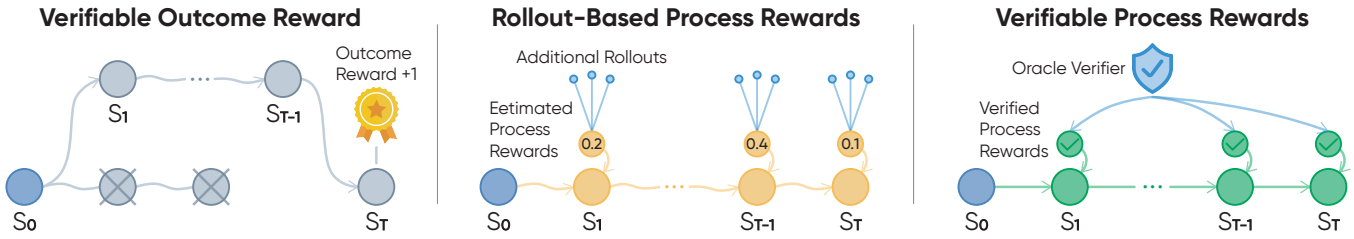

Figure 1: Three reward designs for long-horizon reasoning. Left: outcome-level reward (OR) only fires at trajectory end, leaving intermediate decisions uncredited. Middle: rollout-based process rewards score each step via additional policy rollouts, providing dense feedback but with finite-sample noise (yellow). Right: *Verifiable Process Rewards* (VPR) score each step against a task-specific oracle verifier, producing dense and noise-free turn-level supervision (green).

or vulnerable to reward hacking [5, 38], while rollout-based estimates can be computationally expensive and high-variance as they require sampling multiple completions per state for value estimations [9, 28]. As a result, dense feedback alone is not sufficient: for process rewards to improve long-horizon reasoning, they must also be reliable and objectively grounded.

In this work, we study a class of *densely-verifiable agentic reasoning problems*, where intermediate actions can be objectively checked by symbolic or algorithmic oracles. Such settings arise when the task has explicit structure: search algorithms can verify strategic decisions in dynamic environments, constraint solvers can verify consistency in logical reasoning tasks, and inference engines can verify decisions under uncertainty. These verifiers make it possible to move beyond sparse outcome rewards and construct dense, turn-level supervision that remains objective and grounded.

We propose *Verifiable Process Rewards* (VPR), a framework that converts symbolic or algorithmic oracles into turn-level reward signals for reinforcement learning. Figure 1 contrasts VPR with outcome-level rewards and rollout-based process rewards: instead of waiting for a sparse trajectory-level signal, or relying on noisy rollout estimates, VPR checks each intermediate action against a task-specific verifier and returns a dense, noise-free reward whenever the action is valid or optimal under that verifier. We instantiate VPR in three representative forms of agentic reasoning: search-based verification for *dynamic deduction*, instantiated with Monte Carlo Tree Search (MCTS) [11] to evaluate strategic optimality; constraint-based verification for *logical reasoning*, which checks whether an action remains consistent with the global solution space; and posterior-based verification for *probabilistic inference* [8], which evaluates whether an action is optimal under the current belief state. We complement these instantiations with a theoretical analysis explaining why dense verifiable feedback improves credit assignment. Since each turn carries its own oracle-grounded signal, VPR localizes the policy-gradient update, controls bias through verifier reliability, and can yield more favorable horizon scaling than outcome-level rewards.

We evaluate VPR in controlled densely-verifiable environments designed to isolate three core reasoning abilities: *Tic-Tac-Toe* for dynamic deduction, *Sudoku* for logical reasoning, and *Minesweeper* for probabilistic inference. Across these environments, VPR outperforms outcome-level RL and rollout-based process reward baselines, demonstrating the benefit of reliable turn-level supervision. Importantly, models trained with VPR also improve on general reasoning benchmarks and agentic reasoning tasks, suggesting that verifiable process supervision in densely-verifiable reasoning tasks can foster general reasoning capabilities beyond the training environments. We further analyze training dynamics and oracle quality, showing that VPR leads to more stable learning and that weaker verifiers substantially reduce performance.

Overall, our results suggest that densely-verifiable agentic reasoning provides a useful path for studying how dense, objective process feedback can improve the general reasoning abilities of LLM agents. At the same time, VPR depends on the availability and quality of intermediate verifiers, and extending it to less structured, open-ended environments remains an important challenge. Our contributions are summarized as follows:

- We introduce *Verifiable Process Rewards* (VPR), a framework for deriving process rewards from symbolic or algorithmic verifiers in densely-verifiable agentic reasoning problems.
- We instantiate VPR in three reasoning settings: search-based verification for dynamic deduction, constraint-based verification for logical reasoning, and posterior-based verification for probabilistic inference.
- We provide a theoretical analysis giving an intuition for why dense verifiable feedback improves long-horizon credit assignment, and showing that the verifier-induced gradient bias scales linearly with verifier disagreement and that dense rewards have more favorable horizon scaling than outcome-level rewards.
- We empirically show that VPR outperforms outcome-level RL and rollout-based process reward baselines in controlled densely-verifiable environments, while also improving transfer to general and agentic reasoning benchmarks.

2 METHOD

In this section, we present *Verifiable Process Rewards* (VPR), a framework for converting symbolic or algorithmic verifiers into dense turn-level reward signals for reinforcement learning. We first formalize densely-verifiable agentic reasoning, then describe three concrete VPR instantiations, introduce the turn-level policy optimization objective, and conclude with a brief theoretical analysis.

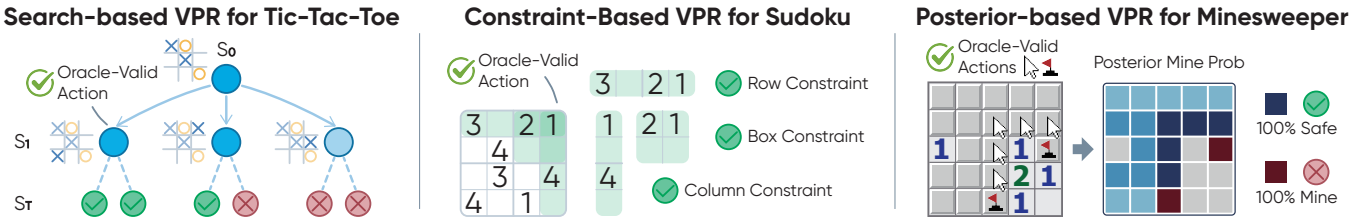

Figure 2: Three VPR instantiations. Search-based (*Tic-Tac-Toe*): MCTS lookahead labels the move with the highest value as oracle-valid. Constraint-based (*Sudoku*): a constraint solver verifies the candidate digit against the row, column, and the local box. Posterior-based (*Minesweeper*): posterior mine probabilities mark zero-probability cells as safe reveals and probability-one cells as flags.

2.1 Densely-Verifiable Agentic Reasoning

We model an episodic agentic reasoning problem as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, H)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition function, R the task reward, and H the horizon. A policy $\pi_\theta(a_t \mid s_t)$ parameterized by an LLM interacts with the environment by producing an action a_t at each state s_t, generating a trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ with $T \le H$. In standard RL from outcome-level verifiable rewards (OR), the reward is sparse and typically nonzero only at the terminal step:

$$r_t^{\mathrm{OR}} = 0 \quad (t < T), \qquad r_T^{\mathrm{OR}} = \mathbb{I}(\text{success}). \tag{1}$$

While objective, this signal provides little information about which intermediate actions caused success or failure.

We focus on a class of *densely-verifiable* agentic reasoning problems, where every intermediate action can be checked by a task-specific verifier $\mathcal{V} : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$, defining the oracle-valid set $\mathcal{A}_{\mathcal{V}}(s) = \{a \in \mathcal{A} : \mathcal{V}(s, a) = 1\}$. VPR converts this verifier into a dense turn-level reward

$$r_t^{\mathrm{VPR}} = \mathcal{V}(s_t, a_t), \tag{2}$$

providing direct feedback on whether each action is valid, useful, or optimal under the task structure.

2.2 Three Instantiations of VPR

The key idea of VPR is to replace heuristic or learned step-level scoring with objective verification whenever the task structure permits. Rather than asking whether an intermediate action *appears* reasonable, VPR checks whether the action satisfies a oracle criterion derived from the task itself. We instantiate this idea in three representative reasoning settings (Figure 2).

Search-Based VPR for Dynamic Deduction. For environments whose states evolve over time, the agent must reason about long-term consequences and avoid locally appealing but strategically losing moves. We use search-based verification with Monte Carlo Tree Search (MCTS) [11], instantiated in *Tic-Tac-Toe*. Letting $Q_{\mathrm{MCTS}}(s, a)$ denote the MCTS value estimate for action a at state s, the oracle-valid set is $\mathcal{A}_{\mathcal{V}}(s) = \arg\max_{a \in \mathcal{A}(s)} Q_{\mathrm{MCTS}}(s, a)$ and $r_t^{\mathrm{VPR}} = \mathbb{I}(a_t \in \mathcal{A}_{\mathcal{V}}(s_t))$. This rewards strategically optimal moves verified by lookahead search.

Constraint-Based VPR for Logical Reasoning. For environments governed by strict symbolic constraints, the agent must keep each local action globally consistent with the eventual solution. We instantiate this in *Sudoku*: for puzzles with a unique solution grid $G^\star$, an action $a_t = (i, j, d)$ fills digit d into cell (i, j), and the verifier checks consistency with the solution: $\mathcal{V}(s_t, a_t) = \mathbb{I}(G^\star[i, j] = d)$. The resulting reward $r_t^{\mathrm{VPR}} = \mathbb{I}(G^\star[i, j] = d)$ provides dense supervision for constraint satisfaction, rewarding local decisions consistent with the global solution.

Posterior-Based VPR for Probabilistic Inference. For partially observable environments, the agent must reason under uncertainty. We instantiate this in *Minesweeper*. Given a board state s_t, let $\Omega(s_t)$ be the set of hidden mine configurations consistent with the revealed observations, and define the posterior probability that cell (i, j) contains a mine,

$$P(\mathrm{mine}_{i,j} \mid s_t) = \frac{\sum_{\omega \in \Omega(s_t)} \mathbb{I}((i, j) \text{ is a mine in } \omega)}{|\Omega(s_t)|}. \tag{3}$$

The agent may either reveal a cell or flag a mine. The verifier sets $\mathcal{V}(s_t, a_t) = 1$ if (i) a_t reveals an unrevealed cell with minimum posterior mine probability (one-step risk-minimizing under the current belief, even when this minimum is positive), or (ii) a_t flags a cell with posterior mine probability 1, with ties treated as oracle-valid. This encourages the policy to update its belief state and act according to posterior uncertainty.

2.3 Turn-Level Policy Optimization

We optimize the policy with a turn-level variant of Group Relative Policy Optimization (GRPO) [21]. For each environment instance q, we sample a group of K trajectories $\{\tau_i\}_{i=1}^K$ from the old policy $\pi_{\theta_{\mathrm{old}}}$ and collect turn-level VPR rewards $r_{i,t}^{\mathrm{VPR}} = \mathcal{V}(s_{i,t}, a_{i,t})$. For each turn t, let $\mathcal{I}_t = \{i : t \le T_i\}$ be the set of trajectories still active at that turn. We normalize rewards across the group to obtain a turn-level advantage,

$$A_{i,t} = \frac{r_{i,t}^{\mathrm{VPR}} - \mu_t}{\sigma_t + \delta}, \qquad \mu_t = \frac{1}{|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} r_{i,t}^{\mathrm{VPR}}, \qquad \sigma_t = \sqrt{\frac{1}{|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} (r_{i,t}^{\mathrm{VPR}} - \mu_t)^2}, \tag{4}$$

and plug $A_{i,t}$ into the standard PPO clipped surrogate

$$J_{\mathrm{VPR}}(\theta) = \mathbb{E}_q \left[\frac{1}{K} \sum_{i=1}^{K} \sum_{t=1}^{T_i} \min\Big(\rho_{i,t}(\theta) A_{i,t},\ \mathrm{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big) A_{i,t} \Big) \right], \tag{5}$$

with importance ratio $\rho_{i,t}(\theta) = \pi_\theta(a_{i,t} \mid s_{i,t})/\pi_{\theta_{\text{old}}}(a_{i,t} \mid s_{i,t})$. Unlike outcome-level RL, each intermediate decision receives its own verifier-derived advantage: correct steps can be reinforced even when the trajectory eventually fails, and invalid steps penalized even when it succeeds by chance.

2.4 Theoretical Analysis

We summarize three results that clarify why and when VPR improves credit assignment. They are first-order, idealized analyses of an unclipped turn-level objective; finite-sample group normalization and PPO clipping are used in practice for stable optimization. Formal statements and proofs are deferred to Appendix C–E.

Proposition 1 (VPR as a local weighted imitation-like update). Consider a fixed state distribution $d(s)$ collected by $\pi_{\theta_{\text{old}}}$ and held independent of θ. Suppose the verifier is aligned with the optimal action set, $\mathcal{V}(s,a) = \mathbb{I}(a \in \mathcal{A}_{\mathcal{V}^\star}(s))$. Then the verifier objective $J_{\mathcal{V}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}[\mathcal{V}(s,a)]$ has policy gradient

$$\nabla_\theta J_{\mathcal{V}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}[\mathcal{V}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)]\,, \tag{6}$$

which is invariant to action-independent baselines. Evaluated at $\theta = \theta_{\text{old}}$, this gradient also equals the gradient of a weighted imitation-like objective that upweights oracle-valid sampled actions, so VPR admits a first-order interpretation as on-policy filtered imitation: every step contributes its own oracle-grounded credit signal, in contrast to outcome-level rewards.

Proposition 2 (Bias scales linearly with verifier error). Consider the idealized per-turn verifier objective under a fixed, θ-independent state distribution $d(s)$. If an approximate verifier $\widehat{\mathcal{V}}$ disagrees with the oracle $\mathcal{V}^\star(s,a) = \mathbb{I}(a \in \mathcal{A}_{\mathcal{V}^\star}(s))$ on a fraction $\bar{\epsilon} = \mathbb{E}_{s\sim d, a\sim\pi_\theta}[\mathbb{I}\{\widehat{\mathcal{V}} \neq \mathcal{V}^\star\}]$ of state–action pairs and $\|\nabla_\theta \log \pi_\theta(a \mid s)\| \le G$ almost surely, then the gradient bias satisfies

$$\left\|\widehat{g}(\theta) - g^\star(\theta)\right\| \le G\bar{\epsilon}. \tag{7}$$

Proposition 1 assumes a perfect verifier; Proposition 2 quantifies what happens when it is approximate. Because the policy-gradient bias scales *linearly* in the verifier disagreement rate $\bar{\epsilon}$, oracle error propagates one-to-one into the gradient, with no horizon-dependent amplification. This favors verifiable process rewards (MCTS / constraint solver / posterior oracles, where $\bar{\epsilon}$ can be driven near zero) over learned or rollout-based process rewards, whose non-trivial $\bar{\epsilon}$ from finite-sample noise or judge bias is inherited at every gradient step.

Proposition 3 (VPR signal accumulates, OR signal is diluted). With score function $\phi_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ and $p_t = \mathbb{E}[\mathcal{V}_t \mid s_t]$, consider the trajectory-level gradient estimators $\widehat{g}^{\text{VPR}} = \sum_t (\mathcal{V}_t - p_t)\phi_t$ (per-step verifier reward) and $\widehat{g}^{\text{OR}} = (\mathbb{I}(\text{succ}) - V^{\text{OR}})\sum_t \phi_t$ (trajectory-level success with scalar value baseline $V^{\text{OR}} = \mathbb{E}[\mathbb{I}(\text{succ})]$). Each per-step expected contribution decomposes as

$$\begin{aligned} \mathbb{E}[(\mathcal{V}_t - p_t)\phi_t] &= \mathbb{E}_{s_t}[\nabla_\theta p_t]\,, \\ \mathbb{E}\big[(\mathbb{I}(\text{succ}) - V^{\text{OR}})\phi_t\big] &= \text{Cov}(\mathbb{I}(\text{succ}), \phi_t)\,. \end{aligned} \tag{8}$$

Even with a perfect verifier, OR and VPR differ in how their gradient signal scales with horizon. Intuitively, the VPR contribution fires at every step regardless of the trajectory's eventual outcome, whereas the OR contribution requires success to be linkable back to step t—an event that becomes exponentially rare when success demands every step be correct. Concretely, in a controlled one-parameter Bernoulli regime with coherent (shared-logit) per-step gradients, where $\mathbb{I}(\text{succ}) = \prod_{t=1}^{T} \mathbb{I}_t$ and each step is correct independently with probability $p \in (0,1)$, aggregating over T steps gives

$$\left\|\mathbb{E}[\widehat{g}^{\text{VPR}}]\right\| = \Theta(T), \qquad \left\|\mathbb{E}[\widehat{g}^{\text{OR}}]\right\| = \Theta(T\,p^T) \xrightarrow{T\to\infty} 0, \tag{9}$$

so the VPR signal grows linearly in horizon while the OR signal is diluted exponentially.

Discussion. Proposition 3 captures the credit-assignment advantage of dense process rewards as a signal-magnitude gap: VPR's per-step contribution is the local verifier gradient at s_t, so the trajectory-level signal accumulates linearly in T, whereas the OR contribution is a single trajectory–score covariance that is diluted when success is the conjunction of many correct steps and, in the multiplicative regime above, collapses exponentially in T while the VPR signal continues to grow. Together, the three propositions explain VPR's qualitative benefit while highlighting its dependence on *oracle quality*—motivating our ablation in Section 3.4—and are first-order interpretations of GRPO, with finite-sample group normalization and PPO clipping adding further effects in practice.

3 EXPERIMENTS

We empirically evaluate the proposed **VPR** framework. Our goal is to understand whether verifiable process supervision can improve multi-turn reasoning, whether such improvements transfer beyond the training environments, and how sensitive the method is to the quality of the verifier. We organize our evaluation around three research questions: (**RQ1, in-domain efficacy**) can VPR improve domain-specific multi-turn reasoning compared with sparse outcome rewards and Monte Carlo process-reward baselines; (**RQ2, out-of-domain generalization**) do reasoning skills acquired in verifiable game environments transfer to general reasoning benchmarks and agentic decision-making tasks; and (**RQ3, oracle quality**) how does the quality of the process oracle affect performance?

3.1 Experimental Setup

Base Model and Training. We use Qwen3-4B [33] with thinking mode turned on as the base model in all experiments across multiple environments, baselines, and ablation settings. All models are trained with a turn-level GRPO objective for 100 update steps with a group size of 128 trajectories per step. Full hyperparameters are reported in Appendix F.

Training Environments. We instantiate VPR in three verifiable multi-turn environments. ***Tic-Tac-Toe*** (dynamic deduction): a compact testbed where optimal play requires tracking the board, anticipating future threats, and avoiding locally appealing but losing moves. During training the agent interacts with a mixed population of MCTS and random opponents to ensure diverse trajectory coverage; for evaluation we play a fixed strong MCTS opponent as both the first (1st) mover and second (2nd) mover. The VPR oracle uses N=10,000 MCTS simulations per move by default. ***Sudoku*** (logical reasoning): 9×9 uniquely-solvable puzzles with 40 blanks, where each action fills one cell and a single invalid assignment can make the remaining trajectory unsolvable. We report success rate (SR, fraction of fully solved puzzles) and completion rate (CR, fraction of correctly filled cells). ***Minesweeper*** (probabilistic inference): a 5×5 grid with 5 hidden mines, where the agent must infer safe moves

Table 1: In-domain performance comparison across the three training environments. Results are mean ± std over 3 evaluation runs, each of 100 games. *Optimal* (gray) denotes the theoretical upper bound; VPR (blue) consistently outperforms the Base model as well as the OR and MC-PR baselines. *Tic-Tac-Toe* reports the average return (optimum 0) when playing first / second against a strong MCTS opponent; *Sudoku* and *Minesweeper* report success rate (SR) and completion rate (CR).

Method	Tic-Tac-Toe		Sudoku		Minesweeper	
	1st	2nd	SR (%)	CR (%)	SR (%)	CR (%)
Optimal	*0.00 ± 0.00*	*0.00 ± 0.00*	*100.00 ± 0.00*	*100.00 ± 0.00*	*100.00 ± 0.00*	*100.00 ± 0.00*
Base	-0.31 ± 0.04	-0.35 ± 0.05	3.91 ± 1.35	63.24 ± 1.72	0.78 ± 0.78	73.71 ± 1.46
OR	-0.18 ± 0.03	-0.21 ± 0.04	48.44 ± 2.61	82.80 ± 1.18	3.91 ± 1.52	77.26 ± 1.31
MC-PR	-0.11 ± 0.04	-0.20 ± 0.05	34.73 ± 2.35	77.39 ± 1.47	2.34 ± 1.11	78.67 ± 1.22
VPR (Ours)	**-0.09 ± 0.03**	**-0.11 ± 0.03**	**56.25 ± 2.28**	**85.13 ± 0.96**	**10.39 ± 1.86**	**80.27 ± 1.08**

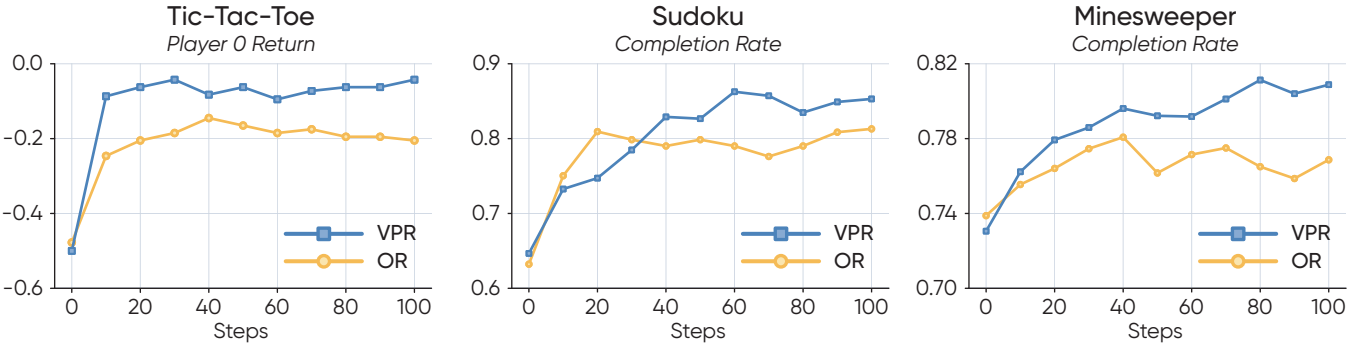

Figure 3: Evaluation curves over GRPO training in the three in-domain environments. VPR (blue) consistently reaches higher final performance than the OR baseline within the same training budget, indicating that dense verifiable feedback improves both sample efficiency and final policy quality.

and mine locations under partial observability. We again report SR and CR. Full evaluation details are reported in Appendix G.

Baselines. We compare VPR against two process-supervision and reinforcement-learning baselines. **OR** provides only sparse trajectory-level rewards; this baseline tests whether final-outcome supervision alone can solve credit assignment in long-horizon reasoning. **MC-PR** estimates intermediate state values using 100 lightweight Monte Carlo rollouts with the policy model under non-thinking mode, and defines the process reward as the temporal difference between consecutive state values. This provides denser feedback than OR but its signal can be noisy as the computational cost of MC rollouts limits the number of simulations.

3.2 In-Domain Performance

Quantitative Results. Table 1 reports in-domain performance across the three training environments. VPR consistently achieves the best result on all six metrics, demonstrating the benefit of verifiable process supervision. In *Tic-Tac-Toe*, VPR approaches the optimal return of 0 and is the only method strong as both first and second player; MC-PR matches VPR as first mover but lags noticeably as second, where dense turn-level credit appears especially helpful. In *Sudoku*, the base model has a moderate completion rate but solves almost no puzzles, showing that locally plausible moves do not by themselves yield globally consistent solutions; MC-PR even underperforms OR, indicating that noisy step-level

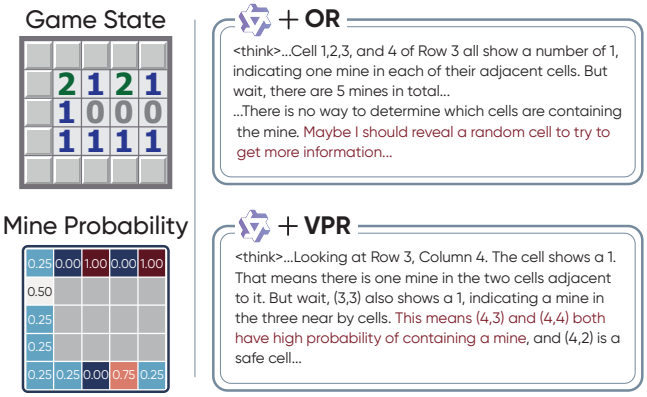

Figure 4: Comparison of VPR and outcome reward (OR) on a representative *Minesweeper* trajectory.

estimates can be worse than sparse outcome supervision in strict constraint-satisfaction settings. *Minesweeper* is the hardest environment, requiring reasoning under partial observability; VPR's larger CR gain shows that its agents make more valid local deductions and survive longer before encountering uncertain states than the

Table 2: Zero-shot transfer to general reasoning benchmarks. We compare the Base model against OR, MC-PR, and VPR (blue) trained in each densely-verifiable environment. Results are mean ± std of pass@1 over n evaluation runs for each benchmark. VPR yields the highest average score for every training environment. Bold marks the best and underline the second-best entry.

Training Env.	Method	GSM8K $n=5$	MATH-500 $n=10$	AIME24 $n=10$	AIME25 $n=10$	GPQA-D $n=10$	BBH $n=1$	MMLU-P $n=1$	Avg.
N/A	Base	94.57 ± 0.13	84.40 ± 1.83	30.00 ± 7.20	18.33 ± 3.93	43.13 ± 3.04	88.39	67.61	60.92
Tic-Tac-Toe	OR	94.53 ± 0.20	84.28 ± 1.11	31.00 ± 5.89	18.67 ± 3.58	43.84 ± 2.29	88.47	67.70	61.21 +0.29
	MC-PR	94.63 ± 0.18	84.36 ± 0.95	30.67 ± 5.16	19.00 ± 3.87	44.19 ± 1.86	88.51	67.72	61.30 +0.38
	VPR	94.74 ± 0.29	83.80 ± 0.35	**33.33 ± 4.97**	**21.33 ± 5.26**	45.15 ± 0.59	**88.96**	67.86	62.17 +1.25
Sudoku	OR	94.40 ± 0.24	84.16 ± 1.21	30.67 ± 5.40	18.67 ± 3.91	45.25 ± 2.43	88.56	67.76	61.35 +0.43
	MC-PR	94.54 ± 0.16	84.28 ± 1.04	30.33 ± 5.08	18.67 ± 3.58	44.04 ± 2.10	88.63	67.65	61.16 +0.24
	VPR	94.65 ± 0.19	83.54 ± 0.61	31.67 ± 4.51	19.67 ± 2.92	**50.20 ± 3.29**	88.82	67.88	62.35 +1.43
Minesweeper	OR	94.56 ± 0.22	84.62 ± 1.31	31.33 ± 5.26	19.00 ± 4.17	44.29 ± 2.03	88.45	67.88	61.45 +0.53
	MC-PR	94.66 ± 0.19	84.58 ± 1.14	31.00 ± 4.98	18.67 ± 3.58	44.55 ± 1.93	88.42	67.73	61.37 +0.45
	VPR	**94.81 ± 0.31**	**85.00 ± 1.06**	32.67 ± 5.40	21.00 ± 3.87	48.33 ± 1.62	88.34	**67.98**	**62.59** +1.67

Table 3: Zero-shot transfer to agentic reasoning tasks. We compare the Base model against OR, MC-PR, and VPR (blue) trained in each densely-verifiable environment, evaluated on ALFWorld (success rate) and WebShop (task score and success rate). Results are mean ± std over $n = 3$ evaluation runs. VPR improves over Base and outperforms OR / MC-PR.

Training Env.	Method	ALFWorld $n=3$ SR (%)	WebShop $n=3$ Score	WebShop $n=3$ SR (%)
N/A	Base	24.13 ± 2.40	27.42 ± 1.00	1.40 ± 0.20
Tic-Tac-Toe	OR	25.37 ± 2.59 +1.24	28.76 ± 1.12 +1.34	1.53 ± 0.31 +0.13
	MC-PR	26.12 ± 2.59 +1.99	29.45 ± 0.98 +2.03	1.67 ± 0.42 +0.27
	VPR	27.36 ± 2.62 +3.23	30.88 ± 0.75 +3.46	1.87 ± 0.50 +0.47
Sudoku	OR	24.88 ± 2.83 +0.75	30.62 ± 1.41 +3.20	1.67 ± 0.31 +0.27
	MC-PR	25.12 ± 2.83 +0.99	30.18 ± 1.53 +2.76	1.73 ± 0.50 +0.33
	VPR	25.62 ± 3.11 +1.49	**34.29 ± 1.86** +6.87	**2.20 ± 0.40** +0.80
Minesweeper	OR	26.12 ± 2.59 +1.99	28.91 ± 1.08 +1.49	1.60 ± 0.40 +0.20
	MC-PR	27.11 ± 2.40 +2.98	29.62 ± 1.26 +2.20	1.73 ± 0.46 +0.33
	VPR	**28.61 ± 2.28** +4.48	30.38 ± 1.20 +2.96	1.93 ± 0.46 +0.53

baselines. Across all three environments, the consistent VPR advantage demonstrates the robustness of dense, noise-free verifiable supervision under diverse reasoning regimes.

Pattern Analysis. A side-by-side trajectory comparison on *Minesweeper* (Figure 4) makes the qualitative difference concrete: the OR-trained policy receives no learning signal until the trajectory terminates, so locally hazardous reveals are not penalized and locally cautious flags are not reinforced; in contrast, VPR scores every intermediate action against the posterior verifier, so risky reveals on high-probability mines incur immediate negative advantage and correct flags receive immediate positive advantage. This per-step credit pattern is what drives VPR's larger CR gain over OR/MC-PR, and is consistent with the signal-magnitude analysis in Proposition 3.

3.3 Out-of-Domain Generalization

We evaluate whether the reasoning skills learned from verifiable game tasks are generalizable to tasks out side the training distribution. We consider 7 general reasoning benchmarks including GSM8K, MATH-500, AIME24/25, GPQA-Diamond, BBH, and MMLU-Pro (Table 2) and 2 agentic reasoning tasks including ALFWorld [24] and WebShop [34] (Table 3) and report the standard pass@1 measured over multiple evaluation runs; no further finetuning is performed.

General Reasoning Benchmarks. Every VPR-trained model improves the average score over the base across all 7 benchmarks, with *Minesweeper*-trained VPR yielding the highest average. The improvements are most visible on harder benchmarks (AIME24/25, GPQA-Diamond) and small or absent on the easiest ones, suggesting that VPR primarily strengthens difficult multi-step reasoning rather than uniformly boosting all tasks. Among the three training environments, *Sudoku*-trained VPR shows the largest gain on

Table 4: Out-of-domain sensitivity to MCTS oracle quality. Same setup as Table 5, but evaluating zero-shot transfer to general reasoning benchmarks; the default N=10,000 row (blue) reproduces the *Tic-Tac-Toe* VPR row of Table 2. The weak N=100 oracle degrades every downstream benchmark below the Base model, while N=1000 recovers most of the benefit, showing that low-quality verifiers harm OOD generalization rather than merely in-domain performance.

Setting	GSM8K $n=5$	MATH-500 $n=10$	AIME24 $n=10$	AIME25 $n=10$	GPQA-D $n=10$	BBH $n=1$	MMLU-P $n=1$	Avg.
Base	94.57 ± 0.13	**84.40 ± 1.83**	30.00 ± 7.20	18.33 ± 3.93	43.13 ± 3.04	88.39	67.61	60.92
$N = 100$	93.84 ± 0.31	82.10 ± 1.95	24.67 ± 6.13	14.33 ± 4.17	40.25 ± 2.87	87.21	66.92	58.47
$N = 1000$	94.66 ± 0.24	83.68 ± 0.48	32.67 ± 4.92	20.67 ± 4.66	44.70 ± 0.93	88.82	67.79	61.86
$N = 10{,}000$ **(Default)**	**94.74 ± 0.29**	83.80 ± 0.35	**33.33 ± 4.97**	**21.33 ± 5.26**	**45.15 ± 0.59**	**88.96**	**67.86**	**62.17**

Table 5: In-domain sensitivity to MCTS oracle quality on *Tic-Tac-Toe*. We vary the number of MCTS simulations N used by the VPR verifier and compare against the Base model. The weakest oracle (N=100) is actively harmful (worse than Base), while the default N=10,000 (blue) is best. Bold marks the best and underline the second-best entry in each column.

Setting	Tic-Tac-Toe Return 1st	Tic-Tac-Toe Return 2nd
Base	-0.31 ± 0.04	-0.35 ± 0.05
$N = 100$	-0.48 ± 0.06	-0.52 ± 0.07
$N = 1000$	-0.13 ± 0.04	-0.15 ± 0.04
$N = 10{,}000$ **(Default)**	**-0.09 ± 0.03**	**-0.11 ± 0.03**

GPQA-Diamond, where constraint elimination is structurally similar to ruling out distractors among multiple-choice options. Beyond this targeted alignment, no individual training environment dominates everywhere, and OR / MC-PR never match VPR's average on any environment, indicating that the broad gains come from dense verifiable process supervision rather than from specific structural quirks of any one game.

Agentic Tasks. On ALFWorld and WebShop, VPR improves over the base regardless of training environment and consistently outperforms OR and MC-PR. *Minesweeper*-trained VPR is best on ALFWorld, consistent with both tasks involving partial observability and step-by-step information gathering. The fact that the gains transfer to embodied text-based planning (ALFWorld) and goal-directed web interaction (WebShop)—domains structurally far from the synthetic training games—indicates that VPR teaches reasoning skills that are not narrowly tied to the training environment.

3.4 Ablation: Oracle Quality

We study how the quality of the process oracle affects learning by varying the number of MCTS simulations in *Tic-Tac-Toe* ($N \in \{100, 1000, 10000\}$) and measuring both in-domain (Table 5) and OOD performance (Table 4). A weak oracle (N=100) actively harms training in both settings: in-domain returns fall below the base model, and the OOD average also drops below it with degradation across every benchmark. This indicates that if the oracle frequently assigns misleading credit, the model can learn worse strategies than those induced by the pretrained policy, and noisy process supervision does not merely fail to help the training task—it can also damage general reasoning capabilities. A moderately strong oracle (N=1000) recovers most of the benefit, while the default N=10,000 is best in both settings. The takeaway is that process rewards must be *both* dense *and* reliable: dense supervision from a misaligned oracle can be worse than sparse outcome supervision, while high-quality verification enables both in-domain skill acquisition and OOD generalization, consistent with Proposition 2's linear $\bar{\epsilon}$ scaling of gradient bias.

4 RELATED WORK

Reinforcement Learning from Verifiable Rewards. Reinforcement Learning from Verifiable Rewards (RLVR) replaces subjective preference-based supervision [18] with objective signals such as mathematical answers, unit tests, symbolic solvers, or executable feedback [4, 14, 19–21, 26]. Most existing RLVR methods operate at the *outcome* level—rewarding the model only after a final answer or full trajectory—which is effective for single-turn problems but provides limited guidance for long-horizon agentic reasoning, where many intermediate decisions may appear locally plausible yet lead to delayed failure. Our work builds on RLVR but shifts the focus from *verifiable outcomes* to *verifiable processes*: search algorithms, constraint solvers, and inference engines supervise intermediate actions, providing dense process-level rewards while preserving the objectivity.

Process Reward Models. Process Reward Models (PRMs) address outcome sparsity by assigning rewards to intermediate steps [15, 28], and fall into two families. Annotation-based PRMs rely on humans or strong LLMs to judge step correctness [3, 5, 30], but inherit annotator cost, inconsistency, and vulnerability to reward hacking. Rollout-based PRMs estimate intermediate values from Monte Carlo rollouts or beam search with the model itself [9, 37], avoiding manual labels but incurring high compute and statistical noise. VPR instead obtains process rewards from task-specific and policy-agnostic oracle verifiers that directly evaluate intermediate actions, retaining PRM-style density while avoiding learned-judge ambiguity and rollout variance. Our oracle-quality ablation further shows that dense supervision is not automatically beneficial: weak verifiers can degrade both in-domain and OOD performance, so VPR additionally emphasizes the reliability and verifiability of the oracle.

LLM Agents and Agentic Reinforcement Learning. LLMs are increasingly used as autonomous agents that interact with tools and environments over multiple turns [27, 31]. Despite rapid

progress on multi-turn benchmarks, agentic RL has largely retained the outcome-only reward structure inherited from RLVR, leaving step-level supervision derived from the environment's structure comparatively under-explored. Inference-time methods such as ReAct [36], Reflexion [22], Tree of Thoughts [35], and LATS [39] enhance planning by reasoning, reflecting, or searching at decoding time, but do not update the underlying policy. More recent work fine-tunes language agents with RL in interactive environments [1, 16], typically using terminal task success as the reward; this black-box formulation is general but ignores the structured nature of many agentic tasks. VPR exploits this structure by converting symbolic verifiers into process-level reward oracles, training agents with dense, objective feedback derived from the environment logic. Compared with annotation- or rollout-based PRMs and with outcome-level agentic RL, VPR thus provides a unified way to learn transferable reasoning skills from verifiable process supervision.

5 CONCLUSION

We presented *Verifiable Process Rewards* (VPR), a framework that turns task-specific verifiers into dense, reliable supervision for intermediate decisions in long-horizon agentic reasoning. Across *Tic-Tac-Toe*, *Sudoku*, and *Minesweeper*, VPR consistently outperforms outcome-reward and Monte Carlo process-reward baselines, and the resulting models transfer to general reasoning benchmarks and agentic tasks such as ALFWorld and WebShop, suggesting that synthetic verifiable environments can serve as useful training grounds for general-purpose multi-turn reasoning.

Our oracle-quality ablation reveals an important caveat: dense feedback is helpful only when it is sufficiently reliable, and weak oracles can degrade both in-domain performance and OOD generalization. VPR thus highlights a practical recipe—identify environments where intermediate correctness can be objectively verified, supervise the reasoning process rather than only the final answer, and transfer the resulting skills to broader agentic settings—and we hope it motivates further work on verifiable environments, stronger process oracles, and methods for extending precise process supervision to less structured real-world tasks.

REFERENCES

[1] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915* (2023).

[2] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-Group Policy Optimization for LLM Agent Training. *arXiv preprint arXiv:2505.10978* (2025).

[3] Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *The Twelfth International Conference on Learning Representations*.

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

[5] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

[6] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720* (2024).

[7] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1 (1998), 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X

[9] Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2025. VinePPO: Refining Credit Assignment in RL Training of LLMs. In *Forty-second International Conference on Machine Learning*.

[10] Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–293.

[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[13] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. 2019. OpenSpiel: A Framework for Reinforcement Learning in Games. *CoRR* abs/1908.09453 (2019). arXiv:1908.09453 [cs.LG] http://arxiv.org/abs/1908.09453

[14] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 21314–21328.

[15] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's Verify Step by Step. In *The Twelfth International Conference on Learning Representations*.

[16] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *The Twelfth International Conference on Learning Representations*.

[17] Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, et al. 2025. GEM: A Gym for Agentic LLMs. *arXiv preprint arXiv:2510.01051* (2025).

[18] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 27730–27744.

[19] Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 3806–3824. https://doi.org/10.18653/v1/2023.findings-emnlp.248

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

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

[22] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 8634–8652.

[23] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053* (2019).

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

[25] ModelScope Team. 2024. EvalScope: Evaluation Framework for Large Models. https://github.com/modelscope/evalscope

[26] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275* (2022).

[27] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[28] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 9426–9439. https://doi.org/10.18653/v1/2024.acl-long.510

[29] Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. 2025. Reinforcement Learning Optimization for Large-Scale Learning: An Efficient and User-Friendly Scaling Library. *arXiv preprint arXiv:2506.06122* (2025).

[30] Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D. Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2024. The Generative AI Paradox: "What It Can Create, It May Not Understand". In *The Twelfth International Conference on Learning Representations*.

[31] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.

[32] Zelai Xu, Zhexuan Xu, Xiangmin Yi, Huining Yuan, Xinlei Chen, Yi Wu, Chao Yu, and Yu Wang. 2025. VS-Bench: Evaluating VLMs for Strategic Reasoning and Decision-Making in Multi-Agent Environments. *coming soon* (2025).

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

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

[35] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 11809–11822.

[36] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[37] Fei Yu, Anningzhe Gao, and Benyou Wang. 2024. OVM, Outcome-supervised Value Models for Planning in Mathematical Reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 858–875. https://doi.org/10.18653/v1/2024.findings-naacl.55

[38] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 46595–46623.

[39] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. In *Forty-first International Conference on Machine Learning*.

A REPRODUCIBILITY STATEMENT

To facilitate future research and ensure the reproducibility of our results, we have made all artifacts publicly available. The source code, model checkpoints, and training scripts utilized in this study can be accessed at https://github.com/thu-nics/VPR. The repository contains comprehensive documentation and configuration files for replicating the experiments in this paper.

B USE OF LLMS

Large Language Models (LLMs) were employed as writing assistants during the preparation of this manuscript. Their usage was exclusively limited to refining grammar, enhancing clarity, and improving overall readability. The core research—including conceptualization, methodology, experimental design, and analysis—remains the original and sole work of the authors.

C PROOF OF PROPOSITION 1

We prove the policy-gradient interpretation of VPR under the idealized setting in Proposition 1. Recall that the verifier objective is

$$J_{\mathcal{V}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_\theta(\cdot|s)}\left[\mathcal{V}(s,a)\right], \tag{10}$$

where $d(s)$ is a fixed state distribution and $\mathcal{V}(s,a)$ is independent of θ. By the log-derivative identity,

$$\begin{aligned}
\nabla_\theta J_{\mathcal{V}}(\theta) &= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a \mid s)\mathcal{V}(s,a) \\
&= \sum_s d(s) \sum_a \mathcal{V}(s,a)\nabla_\theta \pi_\theta(a \mid s) \\
&= \sum_s d(s) \sum_a \pi_\theta(a \mid s)\mathcal{V}(s,a)\nabla_\theta \log \pi_\theta(a \mid s) \\
&= \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}\left[\mathcal{V}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right].
\end{aligned} \tag{11}$$

This establishes the first identity.

Next, for any action-independent baseline $b(s)$, we have

$$\begin{aligned}
\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[b(s)\nabla_\theta \log \pi_\theta(a \mid s)\right] &= b(s) \sum_a \pi_\theta(a \mid s)\nabla_\theta \log \pi_\theta(a \mid s) \\
&= b(s) \sum_a \nabla_\theta \pi_\theta(a \mid s) \\
&= b(s)\nabla_\theta \sum_a \pi_\theta(a \mid s) \\
&= b(s)\nabla_\theta 1 = 0.
\end{aligned} \tag{12}$$

Therefore,

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\left(\mathcal{V}(s,a) - b(s)\right)\nabla_\theta \log \pi_\theta(a \mid s)\right] = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\mathcal{V}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]. \tag{13}$$

Taking expectation over $s \sim d$ gives the same identity for the full expected gradient. This shows that subtracting an action-independent baseline changes variance but not the expected policy gradient.

Finally, consider the weighted imitation-like objective

$$L_{\mathrm{IL}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_{\theta_{\mathrm{old}}}}\left[\mathcal{V}(s,a)\log \pi_\theta(a \mid s)\right]. \tag{14}$$

Its gradient is

$$\nabla_\theta L_{\mathrm{IL}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_{\theta_{\mathrm{old}}}}\left[\mathcal{V}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]. \tag{15}$$

Evaluating this gradient at $\theta = \theta_{\mathrm{old}}$ gives

$$\nabla_\theta L_{\mathrm{IL}}(\theta)|_{\theta=\theta_{\mathrm{old}}} = \mathbb{E}_{s\sim d,\, a\sim\pi_{\theta_{\mathrm{old}}}}\left[\mathcal{V}(s,a)\ \nabla_\theta \log \pi_\theta(a \mid s)|_{\theta=\theta_{\mathrm{old}}}\right], \tag{16}$$

which matches

$$\nabla_\theta J_{\mathcal{V}}(\theta)|_{\theta=\theta_{\mathrm{old}}} = \mathbb{E}_{s\sim d,\, a\sim\pi_{\theta_{\mathrm{old}}}}\left[\mathcal{V}(s,a)\ \nabla_\theta \log \pi_\theta(a \mid s)|_{\theta=\theta_{\mathrm{old}}}\right], \tag{17}$$

because $J_{\mathcal{V}}$ is on-policy at θ_{old}, where $\pi_\theta = \pi_{\theta_{\mathrm{old}}}$. Thus, around the behavior policy $\pi_{\theta_{\mathrm{old}}}$, the VPR policy-gradient update coincides with the first-order gradient of a weighted imitation-like objective on oracle-valid sampled actions.

This completes the proof.

D PROOF OF PROPOSITION 2

By the policy-gradient identity established in Proposition 1, for any binary verifier $\mathcal{U} \in \{0, 1\}$,

$$\nabla_\theta J_{\mathcal{U}}(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}\left[\mathcal{U}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]. \tag{18}$$

Taking the difference between the approximate-verifier gradient and the oracle-verifier gradient gives

$$\widehat{g}(\theta) - g^\star(\theta) = \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}\left[\left(\widehat{\mathcal{V}}(s,a) - \mathcal{V}^\star(s,a)\right)\nabla_\theta \log \pi_\theta(a \mid s)\right]. \tag{19}$$

Since both $\widehat{\mathcal{V}}$ and $\mathcal{V}^\star$ are binary,

$$\left|\widehat{\mathcal{V}}(s,a) - \mathcal{V}^\star(s,a)\right| = \mathbb{I}\left[\widehat{\mathcal{V}}(s,a) \neq \mathcal{V}^\star(s,a)\right]. \tag{20}$$

Using Jensen's inequality and the bounded-score assumption $\|\nabla_\theta \log \pi_\theta(a \mid s)\| \le G$ almost surely, we obtain

$$\begin{aligned}\left\|\widehat{g}(\theta) - g^\star(\theta)\right\| &\le \mathbb{E}_{s\sim d,\, a\sim\pi_\theta}\left[\left|\widehat{\mathcal{V}}(s,a) - \mathcal{V}^\star(s,a)\right| \cdot \|\nabla_\theta \log \pi_\theta(a \mid s)\|\right] \\ &\le G\mathbb{E}_{s\sim d,\, a\sim\pi_\theta}\left[\mathbb{I}\left[\widehat{\mathcal{V}}(s,a) \neq \mathcal{V}^\star(s,a)\right]\right] \\ &= G\bar{\epsilon}.\end{aligned} \tag{21}$$

This completes the proof.

Remark. The bound is tight up to constants. For example, if the approximate verifier disagrees with the oracle verifier on a measurable set of mass $\bar{\epsilon}$ and the score norm attains G in a coherent direction on that set, then the gradient difference can scale as $G\bar{\epsilon}$. The statement is for the idealized per-turn objective under a fixed state distribution. If one instead studies an unnormalized sum over all timesteps, the corresponding bound may include a horizon-dependent factor.

E PROOF OF PROPOSITION 3

Step 1: Per-step decomposition. With $A_t = \mathcal{V}_t - p_t$ and $p_t = \mathbb{E}[\mathcal{V}_t \mid s_t]$, conditional on s_t we have $\mathbb{E}[A_t \mid s_t] = 0$. The VPR contribution satisfies

$$\begin{aligned}\mathbb{E}[(\mathcal{V}_t - p_t)\phi_t \mid s_t] &= \sum_a \pi_\theta(a \mid s_t)\big(\mathcal{V}(s_t,a) - p_t\big)\nabla_\theta \log \pi_\theta(a \mid s_t) \\ &= \sum_a \mathcal{V}(s_t,a)\nabla_\theta \pi_\theta(a \mid s_t) - p_t\,\nabla_\theta \sum_a \pi_\theta(a \mid s_t) \\ &= \nabla_\theta \sum_a \mathcal{V}(s_t,a)\pi_\theta(a \mid s_t) = \nabla_\theta p_t,\end{aligned} \tag{22}$$

since $\sum_a \pi_\theta(a \mid s_t) \equiv 1$. Taking expectation over s_t yields $\mathbb{E}[(\mathcal{V}_t - p_t)\phi_t] = \mathbb{E}_{s_t}[\nabla_\theta p_t]$.

For OR, the score-function identity gives $\mathbb{E}[\phi_t] = 0$, and V^{OR} is a constant, so

$$\mathbb{E}\big[(\mathbb{I}(\mathrm{succ}) - V^{\mathrm{OR}})\phi_t\big] = \mathbb{E}[\mathbb{I}(\mathrm{succ})\phi_t] = \mathrm{Cov}\big(\mathbb{I}(\mathrm{succ}), \phi_t\big). \tag{23}$$

Step 2: Multiplicative-success toy regime. Consider an episodic setting with horizon T in which each step has a binary action $a_t \in \{0, 1\}$ drawn independently from a Bernoulli policy parameterized by a shared logit $\theta \in \mathbb{R}$, so that $\pi_\theta(a_t = 1 \mid s_t) = p = \sigma(\theta)$ for a fixed $p \in (0, 1)$. Let $\mathcal{V}(s_t, a_t) = a_t$, so the verifier endorses action 1 at every state, and let $\mathbb{I}(\mathrm{succ}) = \prod_{t=1}^T a_t$, so trajectory success requires every step to be correct. Under this logit parameterization the score function is then $\phi_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t) = a_t - p$, with $\mathbb{E}[\phi_t] = 0$ and $\mathbb{E}[\phi_t^2] = p(1-p)$.

VPR signal. Here $A_t = a_t - p = \phi_t$, so

$$\mathbb{E}[A_t\phi_t] = \mathbb{E}[(a_t - p)^2] = p(1-p), \qquad \left\|\mathbb{E}[\widehat{g}^{\mathrm{VPR}}]\right\| = T\,p(1-p) = \Theta(T). \tag{24}$$

OR signal. Using independence of the a_t and $a_t^2 = a_t$,

$$\mathbb{E}\big[\mathbb{I}(\mathrm{succ})\,a_t\big] = \mathbb{E}\Big[a_t \prod_{t'\neq t} a_{t'}\Big] = p \cdot p^{T-1} = p^T, \qquad \mathbb{E}[\mathbb{I}(\mathrm{succ})] \cdot p = p^{T+1}, \tag{25}$$

so

$$\mathrm{Cov}\big(\mathbb{I}(\mathrm{succ}), \phi_t\big) = p^T - p^{T+1} = p^T(1-p), \qquad \left\|\mathbb{E}[\widehat{g}^{\mathrm{OR}}]\right\| = T\,p^T(1-p). \tag{26}$$

Since $T\,p^T \to 0$ as $T \to \infty$ for any fixed $p \in (0, 1)$, the OR signal collapses exponentially in T while the VPR signal grows linearly.

Remark (scope). The toy regime is illustrative rather than fully general: it isolates the multiplicative success structure that long-horizon agentic tasks frequently exhibit (a Sudoku trajectory solves the puzzle only if every fill is consistent with the unique solution; strong Tic-Tac-Toe play against a strong opponent requires avoiding strategically losing moves over the whole trajectory). On tasks without strong multiplicative structure (e.g., where partial credit is intrinsic to success), the OR signal need not collapse, and the VPR advantage manifests as a constant-factor improvement rather than as an exponential signal gap.

F IMPLEMENTATION DETAILS

Framework and Software Stack. Our implementation of the VPR framework is built atop ROLL [29], a robust open-source library designed for post-training Large Language Models (LLMs) via reinforcement learning. We leveraged ROLL's native support for multi-turn trajectory generation to handle complex agentic interactions efficiently. To ensure high computational throughput, the system integrates vLLM [12] for efficient inference during the rollout phase and utilizes Megatron-LM [23] for scalable distributed training. The synthetic reasoning environments were implemented using standard libraries to ensure correctness: GEM [17] and VS-Bench [32] were used for puzzle logic (*Sudoku*/*Minesweeper*), while OpenSpiel [13] provided the game-theoretic backend for adversarial tasks like *Tic-Tac-Toe*.

Training Settings. We employ Qwen3-4B as the base policy model for all reported experiments. Training is conducted in a fully online manner: fresh trajectories are sampled from the current policy and immediately used for gradient updates. Specifically, we use GRPO with 128 trajectories per update step and train all models for 100 RL updates. Note that our use of "group" differs from the standard GRPO setting. In standard language-reasoning GRPO, each group typically consists of multiple responses sampled from the same prompt or initial state. In our setting, the 128 trajectories are sampled from different initial game states and together form a single update batch. We apply group-relative normalization across this batch, rather than within multiple same-state response groups. To avoid degenerate normalization at late turns of variable-length episodes (e.g., when only one trajectory in $\mathcal{I}_t$ remains active and the within-batch standard deviation collapses), whenever $|\mathcal{I}_t| < 4$ we fall back to the global mean and standard deviation computed over all (i, t) pairs in the collected trajectory batch.

Since VPR provides dense turn-level supervision, we set the discount factor to $\gamma = 0$, so that each turn-level advantage depends only on the immediate VPR reward. This design avoids propagating delayed rewards across the trajectory and directly optimizes the verifier-labeled validity of each intermediate action. Importantly, the verifier itself already incorporates task-level structure: MCTS captures long-horizon strategic planning, the Sudoku oracle encodes global consistency, and the Minesweeper posterior captures uncertainty under the current belief state. Thus, immediate VPR rewards still reflect non-myopic reasoning signals.

We disable the KL penalty in all main experiments. For optimization, we use the Adam optimizer [10] with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. We adopt a cosine annealing learning rate schedule with a 5-step warm-up to a peak learning rate of 2×10^{-7} before a gradual decay to 0.

Generation Parameters. During the rollout phase, we employ nucleus sampling to generate diverse reasoning paths, using the model's default thinking-mode sampling configuration: temperature $T = 0.6$, Top-P = 0.99, and Top-K = 100. We adopt these defaults rather than tuning them ourselves so that the rollouts reflect the base model's intended exploration behavior in thinking mode.

Hardware Configuration. All experiments, including training and evaluation, were conducted on a single server node equipped with 8 NVIDIA H100 (80GB) GPUs.

G EVALUATION DETAILS

General Reasoning Benchmarks. For single-turn reasoning benchmarks (GSM8K, MATH-500, AIME24/25, GPQA-Diamond, BBH, and MMLU-Pro), we use EvalScope [25] for standardized testing. All models are evaluated zero-shot to assess their intrinsic generalization. We report Pass@1 accuracy (with standard deviation across multiple runs) under the model's thinking mode, in which the model produces a step-by-step derivation before its final answer; the per-benchmark number of runs is reported in Table 2. Predictions are extracted and compared against the ground truth via exact string matching or numeric equivalence.

Agentic Tasks. For interactive agentic tasks, we adopt verl-agent [2] as the evaluation platform.

- **ALFWorld:** We measure the agent's ability to solve embodied text-command tasks, reporting the mean (and standard deviation) of success rate (SR) over 3 runs on the 134-task validation split, with a budget of 30 steps per episode.
- **WebShop:** We measure the agent's interactive decision-making ability in a simulated e-commerce environment, reporting the mean (and standard deviation) of both average score and SR over 3 runs on the full 500-task test split, under the same 30-step budget per episode.

All agentic evaluations use the standard prompts provided by each benchmark to ensure a fair comparison with the baselines.

H GAME OBSERVATION AND PROMPT

Tic-Tac-Toe. For *Tic-Tac-Toe*, we provide the agent with a complete observation of the 3x3 game board. The state of each cell—whether it is empty, occupied by 'X', or occupied by 'O'—is explicitly provided. The prompt clearly indicates which player's turn it is ('X' or 'O') and presents the current board state, asking the agent to select coordinates for its next move from the available empty cells. For example, the game begins with a prompt that provides the empty 3x3 grid and asks the agent to make the first move (Listing 1).

Listing 1: Prompt for *Tic-Tac-Toe*.

```
system_prompt:
You are an AI agent that makes optimal decisions to win in the game of Tic-Tac-Toe.

user_prompt:
GAME RULES:
```

```
1. Tic-tac-toe is a two-player board game played on a three-by-three grid. The grid is 0-indexed, where
(0,0) is the top-left corner and (2,2) is the bottom-right corner.
2. Two players take turns placing their marks X and O in empty cells of the grid.
3. The player who first places three of their marks in a horizontal, vertical, or diagonal line wins.
4. If all cells are filled and no player wins, the game ends in a draw.

PLAYER INFORMATION:
1. Your mark is X. You are competing with another player controlling the mark O.
2. In each of your turns:
   a. The game state demonstrates the current board with a three-line text grid, where 'X' and 'O' are
   the marks of the two players, and '.' represents empty cells.
   b. You need to choose an action to place your mark in an empty cell, based on the given game state and
    the history of your decisions.
   c. All legal actions for the current turn are provided in the format of `<X({row},{column})>`, where `
   X` is your mark, and {row} and {column} are integers indicating the row and column of the cell to
   place your mark.

RESPONSE INSTRUCTIONS:
Always choose only one action from the legal actions and output `<answer>{your chosen action}</answer>`
with no extra text after you finish the thinking process. For example, `<answer><X(0,0)></answer>`.
Strictly follow the above format and keep your thinking process concise. Responses that do not follow the
 format will result in immediate loss of the game.

The game state is provided below. Please choose your action and strictly follow the given output format
in the response instructions.

GAME STATE:
    0  1  2
 0  .  .  .
 1  .  .  .
 2  .  .  .
```

Sudoku. For *Sudoku*, the agent is presented with a standard 9x9 grid state. The observation uses a text-based matrix where numbers represent pre-filled or agent-filled cells, and '.' denotes empty cells. Rows and columns are explicitly indexed (R1-R9, C1-C9) to facilitate coordinate selection. The prompt outlines the standard constraint satisfaction rules—requiring unique digits 1 through 9 in every row, column, and 3×3 subgrid—and asks the agent to specify a valid move. The action format requires specifying the row, column, and the digit to be placed (Listing 2).

Listing 2: Prompt for *Sudoku*.

```
system_prompt:
You are an AI agent that makes optimal decisions to solve the Sudoku puzzle.

user_prompt:
GAME RULES:
1. Sudoku is played on a 9x9 grid. Rows and columns are 1-indexed (1 to 9).
2. The goal is to fill the empty cells with digits from 1 to 9.
3. Each row must contain all digits from 1 to 9 without repetition.
4. Each column must contain all digits from 1 to 9 without repetition.
5. Each of the nine 3x3 subgrids must contain all digits from 1 to 9 without repetition.
6. You cannot overwrite pre-filled cells.

PLAYER INFORMATION:
1. The current board state is displayed as a text grid.
   - '.' represents an empty cell.
   - Numbers represent filled cells.
   - Rows are labeled R1, R2... and Columns C1, C2...
2. In each turn, you choose an action to fill an empty cell with a number.
3. All legal actions are provided in the format `<fill({row},{col},{number})>`.

RESPONSE INSTRUCTIONS:
```

```
Always choose strictly one action and output `<answer>{your chosen action}</answer>` with no extra text
after you finish the thinking process. For example, to fill row 1, column 1 with number 5, output `<
answer><fill(1,1,5)></answer>`. Strictly follow the above format. Responses that do not follow the format
 will result in penalties.

The game state is provided below. Please choose your action and strictly follow the given output format
in the response instructions.

GAME STATE:
   C1 C2 C3   C4 C5 C6   C7 C8 C9
R1  4  .  . |  9  5  . |  2  .  1
R2  .  .  . |  3  6  . |  .  .  .
R3  .  6  . |  .  8  4 |  9  5  3
   - - - - - - - - - - - - - - - -
R4  .  9  8 |  .  7  5 |  .  .  2
R5  .  .  . |  .  9  3 |  1  .  4
R6  3  7  . |  6  2  . |  .  8  9
   - - - - - - - - - - - - - - - -
R7  .  3  . |  2  4  . |  8  .  .
R8  .  .  6 |  .  1  . |  .  2  5
R9  .  .  . |  5  3  8 |  4  1  .
```

Minesweeper. For *Minesweeper*, the environment consists of a 5×5 grid containing exactly 5 hidden mines. The observation provides the current board state, visually distinguishing between unrevealed cells ('.'), flagged cells ('F'), and revealed safe cells which display the count of adjacent mines (0-8). The prompt instructs the agent to perform probabilistic reasoning to reveal safe cells while avoiding mines. Unlike the other games, the agent has two distinct action types: revealing a cell or toggling a flag on a suspected mine, both formatted as specific command tags (Listing 3).

Listing 3: Prompt for *Minesweeper*.

```
system_prompt:
You are an AI agent that makes optimal decisions to solve the game of Minesweeper.

user_prompt:
GAME RULES:
1. Minesweeper is played on a 5x5 grid of cells. The grid contains exactly 5 hidden mines. The grid is 0-
indexed, where (0,0) is the top-left corner and (4,4) is the bottom-right corner.
2. The goal is to reveal all cells that do not contain mines without revealing any mine.
3. If you reveal a mine, you lose the game immediately.
4. If you reveal a safe cell, it will show a number indicating how many mines are adjacent to it (
neighbors include diagonals).
5. You can also place a flag on a cell if you suspect it contains a mine, or remove a flag if you change
your mind.

PLAYER INFORMATION:
1. The current board state is displayed as a text grid, where:
   - '.' represents an unrevealed cell.
   - 'F' represents a flagged cell.
   - A number (0-8) represents a revealed safe cell with that many adjacent mines.
2. In each turn, you must choose an action to either reveal a cell or flag/unflag a cell.
3. All legal actions are provided in the format `<reveal({row},{col})>` or `<flag({row},{col})>`. The '
flag' command acts as a toggle: play it on an unflagged cell to place a flag, or on a flagged cell to
remove it.

RESPONSE INSTRUCTIONS:
Always choose strictly one action and output `<answer>{your chosen action}</answer>` with no extra text
after you finish the thinking process. For example, to reveal the cell at row 0, column 0, output `<
answer><reveal(0,0)></answer>`. To flag (or unflag) the cell at row 1, column 2, output `<answer><flag
(1,2)></answer>`. Strictly follow the above format. Responses that do not follow the format will result
in immediate loss of the game.
```

```
The game state is provided below. Please choose your action and strictly follow the given output format
in the response instructions.

GAME STATE:
    0  1  2  3  4
 0  .  .  .  .  .
 1  .  .  .  .  .
 2  .  .  .  .  .
 3  .  .  .  .  .
 4  .  .  .  .  .
```