CAST: GAME SOLVERS AS TURN-LEVEL TEACHERS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training large language models (LLMs) to act in long-horizon games is a promising step toward generalist decision-making, yet reinforcement learning with verifiable rewards (RLVR) relies on sparse final rewards that reveal little about which decisions determine success. Denser process signals could supply this missing turn-level credit, but existing sources are hard to keep both cheap and accurate. We observe that changes in a game solver's state value reveal whether an action advances the state toward success. Building on this insight, we propose **CAST** (**C**redit **A**ssignment from **S**olver **T**eachers), which converts these value changes into solver advantages and injects them into RLVR as turn-level signals. We further show that, under a soft-optimal solver assumption, maximizing the solver advantage is equivalent to on-policy distillation from the solver, requiring only scalar values rather than teacher logits. Across Sokoban, Minesweeper, and Rush Hour, CAST outperforms all trained baselines on every game under both in-domain and unseen-difficulty evaluation and achieves the highest average zero-shot performance on ALFWorld and WebShop.

1 INTRODUCTION

As large language models (LLMs) and multimodal foundation models (Yang et al., 2025; Team et al., 2026) move from passive generation toward active decision-making, a central goal is to build generalist agents that can make decisions in embodied and open-world environments (Intelligence et al., 2025). Such agents need to act in evolving states, where the effects of early decisions may emerge only later and are often difficult to reverse (Ye et al., 2026). These settings require a broad range of capabilities, including long-horizon planning, goal-directed exploration, and recovery from mistakes. Classical games such as Sokoban and Minesweeper offer an ideal testbed for these capabilities (Junghanns & Schaeffer, 2001; Mnih et al., 2015; Hu et al., 2025), with well-defined rules, verifiable feedback, and scalable decision-making environments. In fact, such games can already be solved efficiently by domain-specific solvers, ranging from heuristic search and dynamic programming to specialized DQN value networks (Mnih et al., 2013). General-purpose LLMs, by contrast, remain unreliable even in these structured environments, as shown in Figure 1 (left). Games therefore provide a bridge for studying how a general-purpose LLM can become a reliable interactive agent (Ouyang et al., 2026; Paglieri et al., 2025).

A central bottleneck in training generalist LLM game agents is the lack of scalable, fine-grained learning signals. Reinforcement Learning with Verifiable Rewards (RLVR) has been effective in single-turn tasks (Zhang et al., 2026; Chen et al., 2026b; Guo et al., 2025), but in games, final rewards are often too sparse. As illustrated in Figure 1 (right), such outcome signals cannot reveal which decisions in a trajectory cause the final win or loss, creating a fundamental credit-assignment challenge. To address this challenge, existing approaches obtain denser process signals through costly search (Hao et al., 2023), learned process reward models (Xi et al., 2026), or cross-trajectory comparisons such as GiGPO (Feng et al., 2026), yet still face trade-offs among computation, supervision, and signal reliability. We observe that game solvers can provide this missing signal, as a solver can evaluate an action from the state at which it was taken. SFT on solver-generated expert trajectories, however, does not fully exploit this capability: it exposes the LLM only to expert-visited states and offers little guidance once the model deviates during interaction. This motivates an on-policy RL paradigm in which the LLM explores under its own policy and the solver acts as a *turn-level teacher*.

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.

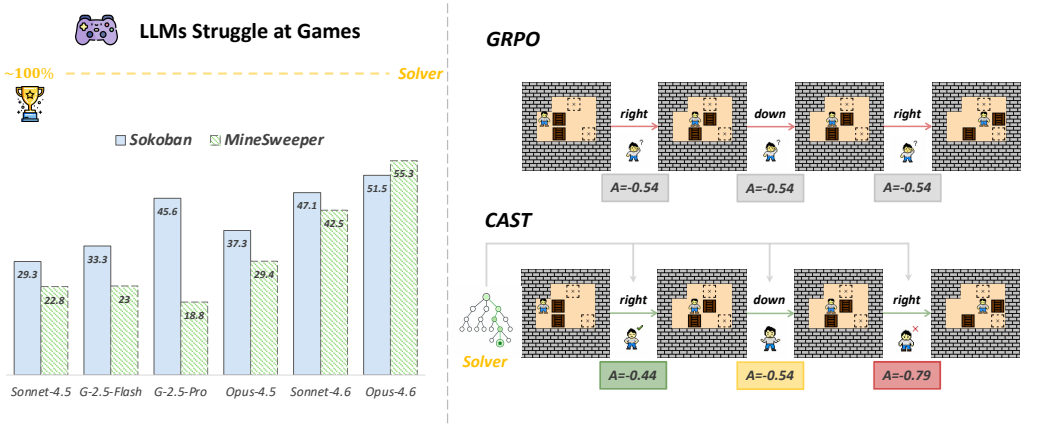


Figure 1: **Overview. (Left)** Several current closed-source LLMs struggle on classical games. **(Right)** Outcome-only RLVR lacks fine-grained feedback, whereas our solver-derived signal provides accurate turn-level credit.

Aligning the model's on-policy exploration with immediate solver feedback is, in essence, a form of on-policy distillation (OPD). Existing LLM distillation methods (Lu & Lab, 2025; Gu et al., 2024), however, typically rely on teacher logits over the full token space, whereas classical solvers return only an optimal action or a scalar cost-to-go, not such a distribution. We therefore propose **CAST** (**C**redit **A**ssignment from **S**olver **T**eachers), a distillation strategy built directly on solvers. Since a solver can complete a game from an intermediate state, it assigns every state a *state value*: how close that state is to winning. For each action sampled by the LLM, CAST compares the solver values immediately before and after the action to obtain a solver advantage, and injects it into RLVR as turn-level signals. This converts the outcome reward into process supervision indicating whether the current action moves the state closer to success, at negligible training overhead. Our theoretical analysis shows that, for a sufficiently strong solver, an action's log-probability under the solver's implicit action distribution is strictly proportional to its one-step change in state value. Consequently, increasing the likelihood of actions with larger solver advantages is mathematically equivalent to on-policy distillation that aligns the LLM's action distribution with the solver's. This equivalence resolves the obstacle above: the scalar advantage already carries the teacher's action preference, so a single scalar suffices for distillation without the teacher's full action distribution, yielding an efficient logit-free distillation.

In practice, the absolute scale of solver signals varies across games and often contains extreme values that destabilize training. We address this with two lightweight shaping steps. First, an `asinh` transformation smoothly compresses extreme values while preserving resolution in the small-signal regime; a theoretical analysis shows this amounts to a robustified KL constraint. Second, batch-level root-mean-square (RMS) normalization removes cross-domain scale differences. These let the solver advantage serve as a stable, comparable process signal that integrates seamlessly into RLVR.

We evaluate CAST on Sokoban, Minesweeper, and Rush Hour, which span complementary challenges in long-horizon planning, partial-observation inference, and constrained combinatorial search. CAST achieves the best performance among all trained methods across every game under both in-domain and unseen-difficulty settings, consistently outperforming outcome-only RLVR baselines and the process-level baseline GiGPO (Feng et al., 2026). It also reaches DAPO's (Yu et al., 2026a) peak validation performance in substantially fewer training steps and transfers zero-shot to held-out ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022) domains without further fine-tuning. Ablations validate the solver-advantage weighting and signal transformations; further analyses show negligible solver overhead and that a learned value network retains much of the benefit of exact solver guidance.

In summary, our contributions are as follows:

- **Solver-derived credit.** We cast game solvers as turn-level teachers and define a solver advantage that assigns fine-grained credit to LLM-sampled actions in RLVR.

- **Logit-free distillation.** We prove that maximizing this advantage is equivalent to on-policy distillation, giving a distillation objective needing no teacher logits, and stabilize it with `asinh` compression and batch-level RMS normalization.
- **Performance and generalization.** CAST reaches state-of-the-art results on all three games in-domain and at unseen difficulties, improves sample efficiency, and transfers zero-shot to ALFWorld and WebShop.
- **Practicality.** An exact solver adds negligible overhead, and a learned value network provides comparable performance when no exact solver exists.

2 METHOD

Our method lets a game-specific solver act as a turn-level teacher that scores each action the LLM takes, refining the sparse terminal reward into a dense per-step process signal. Subsection 2.1 formalizes games as reinforcement learning problems and identifies why outcome-only RLVR is limited. Subsection 2.2 introduces the solver teacher, constructs its per-step score, gives the full pipeline that makes it stable and integrates it into training, and finally explains why this score is equivalent to on-policy distillation (OPD, Theorem 2.1).

2.1 PRELIMINARIES

Games as multi-turn MDPs. We formulate each game as a finite-horizon Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, H)$ with horizon H. At turn t, the LLM policy π_θ observes a textual rendering of the state $s_t \in \mathcal{S}$, samples an action $a_t \sim \pi_\theta(\cdot \mid s_t)$, and the environment transitions via $P(s_{t+1} \mid s_t, a_t)$ until the episode terminates at some step $T \le H$, yielding a trajectory $\tau = (s_0, a_0, \ldots, s_T)$. For Sokoban and Rush Hour, s_t is the complete board; for Minesweeper, s_t is the *information state* (the revealed board plus the constraints it induces on hidden mines), which is a sufficient statistic that keeps the process Markov. The reward is sparse and verifiable:

$$r_t = 0 \quad (t < T), \qquad R(\tau) = r_T = \mathbf{1}\{s_T \in \mathcal{S}_{\text{solved}}\}. \tag{1}$$

This $0/1$ terminal reward is the true optimization objective throughout the paper; every solver signal introduced later is auxiliary supervision and does not change what counts as winning.

Outcome-supervised RLVR and its bottleneck. RLVR maximizes the expected return $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$ via policy gradients (Sutton et al., 1999). To avoid a learned critic, GRPO (Shao et al., 2024) samples a group of G trajectories from $\pi_{\theta_{\text{old}}}$ for a prompt q and normalizes their returns into a trajectory-level advantage,

$$\hat{A}_i^{\text{outcome}} = \frac{R_i - \mu_R}{\sigma_R + \delta}, \qquad \mu_R = \frac{1}{G}\sum_{j=1}^{G} R_j, \quad \sigma_R = \sqrt{\frac{1}{G}\sum_{j=1}^{G}(R_j - \mu_R)^2}, \tag{2}$$

with a small constant $\delta > 0$ for numerical stability. This trajectory-level scalar is assigned to every token of the trajectory and optimized through a clipped surrogate objective, where the importance-sampling ratio ρ_t between π_θ and $\pi_{\theta_{\text{old}}}$ is clipped to $[1-\epsilon, 1+\epsilon]$:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\,\{o_i\}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Big(\min\big(\rho_t\,\hat{A}_i,\ \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\,\hat{A}_i\big) - \beta_{\text{kl}}\,D_{\text{KL}}\big[\pi_\theta \,\|\, \pi_{\text{ref}}\big]\Big)\right]. \tag{3}$$

The bottleneck is now explicit: because $\hat{A}_i = \hat{A}_i^{\text{outcome}}$ is computed solely from the terminal outcome, every turn in a trajectory receives the same credit. This coarse trajectory-level credit is the root of the credit-assignment failure in long-horizon games, motivating the turn-level solver signal introduced next.

2.2 SOLVER-GUIDED TURN-LEVEL CREDIT

A per-step score from the solver. To score each action individually, we need a way to measure how good the current state is. The $0/1$ terminal reward cannot do this, as it only takes a value at the very end. A solver can, because it completes a game from any state. This property equips any solver

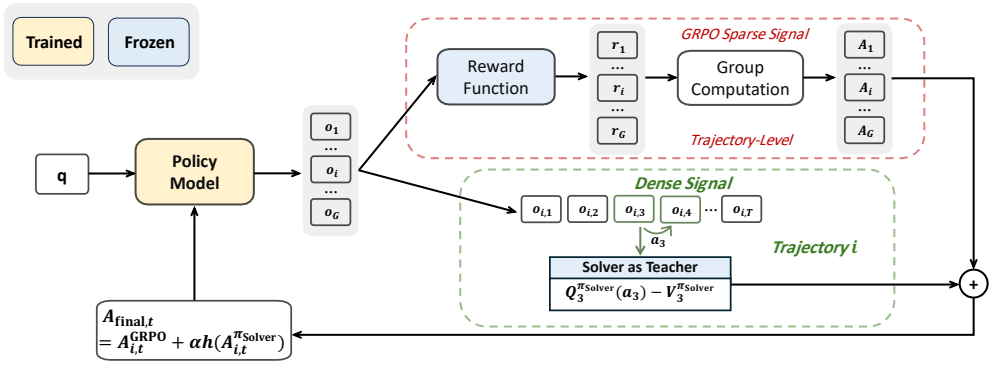


Figure 2: **Method overview.** We augment GRPO's outcome advantage with a shifted solver advantage derived from turn-level cost-to-go changes.

with a general-purpose measure that we call the *cost-to-go* $N(s)$: the minimum amount of work needed to reach a win from state s. Its exact unit varies by game, but the meaning of "how far from the goal" is universal: for Sokoban and Rush Hour, $N(s)$ is the minimum number of actions to solve the board, and for Minesweeper it is the minimum number of reveals needed to clear all safe cells.

A natural per-step score is $N(s_t) - N(s_{t+1})$, the drop in cost-to-go caused by the move. We ground this score in RL by defining an auxiliary shortest-path objective (-1 per action, terminal value 0) and setting $V^{\pi_{\text{Solver}}}(s) = -N(s)$, which makes states closer to the goal more valuable.

$$\begin{aligned} Q^{\pi_{\text{Solver}}}(s_t, a_t) &= -1 + \mathbb{E}_{s_{t+1}}[V^{\pi_{\text{Solver}}}(s_{t+1})], \text{ and } V^{\pi_{\text{Solver}}}(s_t) = -N(s_t) \\ A^{\pi_{\text{Solver}}}(s_t, a_t) &= -1 + N(s_t) - \mathbb{E}_{s_{t+1}}[N(s_{t+1})]. \end{aligned} \tag{4}$$

The advantage $A = Q - V$ measures how much better a specific action is compared to the expected outcome under the solver's policy. The $\mathbb{E}_{s_{t+1}}$ is over the environment transition; for the deterministic games studied here it reduces to the single resulting state, and we keep the expectation for generality.

A shift so that progress maps to positive credit. Under the solver's optimal policy, the expected next-state cost $\mathbb{E}_{s_{t+1}}[N(s_{t+1})]$ equals $N(s_t) - 1$, because the solver always reduces the remaining steps by exactly one. Substituting into Equation 4 gives $A^{\pi_{\text{Solver}}} = -1 + (N(s_t) - (N(s_t) - 1)) = 0$: an optimal action has zero advantage, and any suboptimal action that fails to reduce the cost-to-go by a full step scores negative. This makes the raw advantage non-positive, which is inconvenient as a signal. Shifting by $+1$ gives the *shifted solver advantage*,

$$\widetilde{A}^{\pi_{\text{Solver}}}(s_t, a_t) = A^{\pi_{\text{Solver}}}(s_t, a_t) + 1 = N(s_t) - \mathbb{E}_{s_{t+1}}[N(s_{t+1})], \tag{5}$$

which equals the drop in cost-to-go caused by the move. Its meaning is direct: an optimal move that advances one step receives $+1$, a move with no progress receives 0, and a harmful move receives negative credit. The one special case is a transition into an unsolvable dead state, where $N(s_{t+1}) = \infty$; we cap such a transition at the finite penalty $-N(s_t)$, counting it as losing the $N(s_t)$ steps that would otherwise have won the game.

Shaping the signal for training. The shifted advantage is clean in theory but problematic if used directly. First, it is heavy-tailed: most moves change the cost-to-go by only 0 or ± 1, but the dead-state penalty $-N(s_t)$ can be large, and a few such extremes would dominate the gradient. Second, its scale varies across games, as a hard board has a cost-to-go of dozens while an easy one has only a few. We address both with two lightweight transformations. First, an `asinh` compression,

$$g(x) = \operatorname{asinh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right), \tag{6}$$

which is near-linear for small x, so common small-progress values are preserved and remain distinguishable, and grows only logarithmically for large $|x|$, so rare dead-state penalties are compressed. We use `asinh` rather than a plain logarithm because it is defined and odd-symmetric across negative, zero, and positive values, fully preserving the sign that encodes beneficial, neutral, and harmful moves.

Second, batch-level RMS normalization rescales the signal by its root-mean-square magnitude over all turns in the batch,

$$h(x) = \frac{g(x)}{\mathrm{RMS}_{\mathcal{B}}(g) + \epsilon}, \qquad \mathrm{RMS}_{\mathcal{B}}(g) = \sqrt{\tfrac{1}{|\mathcal{B}|} \textstyle\sum_{(i,t)\in\mathcal{B}} g\Big(\widetilde{A}^{\pi_{\text{Solver}}}(s_{i,t}, a_{i,t})\Big)^2}, \tag{7}$$

where $\mathcal{B}$ contains all turns in the batch, placing the signal on a consistent scale across games. We divide by the RMS *without* subtracting the mean, which is deliberate: the value 0 should keep meaning "no progress" so that positive stays beneficial and negative stays harmful; subtracting the mean would shift this zero point and destroy the sign information on which the signal relies.

Combining with GRPO. We combine this per-step process signal with the terminal reward, so as to keep the true objective of winning while adding fine-grained per-step credit (Figure 2). For trajectory i at turn t, we write $\widetilde{A}^{\pi_{\text{Solver}}}_{i,t} \triangleq \widetilde{A}^{\pi_{\text{Solver}}}(s_{i,t}, a_{i,t})$ for the shifted solver advantage evaluated at that trajectory's state-action pair. We add the shaped solver signal to GRPO's outcome advantage and broadcast the result to every token of the turn,

$$\hat{A}_{i,t} = \hat{A}^{\text{outcome}}_i + \alpha\, h\Big(\widetilde{A}^{\pi_{\text{Solver}}}_{i,t}\Big), \tag{8}$$

with a single coefficient α controlling the strength of solver guidance. Three granularities meet here: $\hat{A}^{\text{outcome}}_i$ is trajectory-level and identical across all turns of a trajectory, anchoring the global win-or-loss credit; $h(\widetilde{A}^{\pi_{\text{Solver}}}_{i,t})$ is turn-level and varies step by step, refining credit within the trajectory; and their sum is broadcast to the token level, since GRPO optimizes over tokens. The combined advantage $\hat{A}_{i,t}$ replaces $\hat{A}^{\text{outcome}}_i$ in the clipped GRPO surrogate of Equation 3, so the terminal reward still anchors trajectory-level credit while the solver signal refines it turn by turn.

Equivalence to logit-free OPD. Having presented the complete pipeline, we now show that the solver advantage is not merely a hand-designed signal but is mathematically equivalent to distilling from the solver. The key observation is that a sufficiently strong solver can be viewed as a *soft-optimal* teacher policy whose action probabilities grow exponentially with the action value: $\pi_{\text{Solver}}(a \mid s) \propto \exp\big(Q^{\pi_{\text{Solver}}}(s,a)/\tau\big)$ for a temperature $\tau > 0$ (Ziebart et al., 2008; Haarnoja et al., 2018). Taking log immediately gives $A^{\pi_{\text{Solver}}}(s,a) = \tau \log \pi_{\text{Solver}}(a \mid s)$: the solver advantage is exactly the teacher's log-preference for the action. We then obtain the following result (proved in Appendix C.5):

Theorem 2.1 (Implicit Objective of CAST)**.** *Assume the solver is soft-optimal with temperature $\tau > 0$, the solver advantages satisfy $|A^{\pi_{Solver}}| \lesssim 1$, and GRPO provides an unbiased task-return gradient under a frozen-visitation surrogate. Then the policy-gradient update of the training rule Equation 8 equals the gradient of*

$$\mathcal{J}(\theta) = \underbrace{\mathbb{E}_{s_0\sim\mu}\big[V^{\pi_\theta}_{task}(s_0)\big]}_{\textit{task return}} - \beta\, \underbrace{\mathbb{E}_{s\sim d^{\pi_\theta}}\Big[\mathrm{H}\big(\pi_\theta(\cdot|s),\, \pi_{Solver}(\cdot|s)\big)\Big]}_{\textit{cross-entropy distillation}}, \quad \beta = \frac{\alpha\,\tau}{\sqrt{2}\,(\mathrm{RMS}_{\mathcal{B}}(g) + \epsilon)}, \tag{9}$$

where $\mathrm{H}(\pi_\theta, \pi_{Solver}) = -\mathbb{E}_{a\sim\pi_\theta}[\log \pi_{Solver}(a \mid s)]$ is the cross-entropy from π_θ to π_{Solver}, and $d^{\pi_\theta}(s)$ is the expected number of times state s is visited during an episode.

The theorem reveals that our method implicitly maximizes the task return while minimizing the cross-entropy from π_θ to π_{Solver}, which is the objective of OPD (Lu & Lab, 2025). We have:

Logit-free OPD. Traditional distillation requires the teacher's full output distribution. Here, the identity $A^{\pi_{\text{Solver}}} = \tau \log \pi_{\text{Solver}}$ means a single scalar per action already encodes the teacher's log-preference, that is, no logits needed.

Why the student can surpass the solver. Isolating the KL component of Equation 9 and solving the per-state optimization yields a closed-form optimal policy $\pi^*(a|s) \propto \pi_{\text{Solver}}(a|s)\, \exp\big(A^{\pi_\theta}_{\text{task}}(s,a)/\beta\big)$: the solver distribution serves as a prior, and the task advantage tilts it exponentially. At $\beta \to \infty$ (pure distillation), $\pi^* \to \pi_{\text{Solver}}$; at finite β, the tilt lets π^* deviate wherever the task reward warrants it, so the student can *surpass* the teacher.

3 EXPERIMENTS

Our experiments address four research questions below:

RQ1: Does the solver-derived turn-level signal outperform outcome-only and process-level RL baselines in-domain on the training games (Subsection 3.2)?

RQ2: Do the trained agents generalize to unseen difficulty levels within the training games and to held-out domains (Subsection 3.2, Subsection 3.3)?

RQ3: Does each design component contribute to the final performance, namely the solver-advantage weight, the `asinh` transformation, and batch-level RMS normalization (Subsection 3.4)?

RQ4: Is solver guidance practical, i.e., does it add little training overhead and remain effective when the solver is a learned value network (item 3.4)?

3.1 EXPERIMENTAL SETUP

Datasets and Benchmarks We train and evaluate on three classic games, **Sokoban**, **Minesweeper**, and **Rush Hour**, whose clear rules and efficient solvers provide verifiable rewards and process-level supervision. Together, they span complementary reasoning challenges, from long-horizon planning and partial-observation inference to constrained combinatorial search. For each game, we procedurally generate instances across difficulty levels defined by game-specific structural parameters and calibrated with solver-derived solution complexity. We canonicalize board configurations to remove duplicates, prevent overlap between training and test instances, and verify solvability with the corresponding solver. Evaluation covers two within-game settings: *In-Domain* (ID), covering difficulty levels seen during training, and *Unseen-Difficulty*, covering harder levels held out from training. We evaluate each setting on 200 held-out instances per game. Throughout, *unseen difficulty* denotes this within-game shift, whereas *OOD transfer* denotes zero-shot evaluation on a held-out task domain. Full generation and difficulty-control details are given in Appendix B.1.

To probe OOD transfer beyond the training games, we further evaluate zero-shot on two held-out agentic domains without any further fine-tuning: the embodied benchmark ALFWorld (Shridhar et al., 2021) and the web benchmark WebShop (Yao et al., 2022); benchmark sources and evaluation protocols are detailed in Appendix B.2.

Baselines and Models We use *Qwen3-4B-Instruct-2507* (Yang et al., 2025) as the base policy for all trained methods. As training-free references, we evaluate the frozen base policy with a ReAct-style prompt (Yao et al., 2023), together with a suite of strong closed-source models under the same prompt, namely *Gemini-2.5-Flash* (Comanici et al., 2025), *Gemini-2.5-Pro* (Comanici et al., 2025), *Claude-Sonnet-4.5* (Anthropic, 2025b), *Claude-Opus-4.5* (Anthropic, 2025a), *Claude-Sonnet-4.6* (Anthropic, 2026b), and *Claude-Opus-4.6* (Anthropic, 2026a). For trained baselines, we compare against outcome-only RLVR methods that fine-tune the same base policy with the same sparse terminal reward: GRPO (Shao et al., 2024), GSPO (Zheng et al., 2025), and DAPO (Yu et al., 2026a). This keeps the terminal reward fixed and focuses the comparison on the added solver-derived turn-level signal. We also include GiGPO (Feng et al., 2026), a process-level baseline, to examine whether solver-guided supervision provides gains beyond anchor-state process shaping.

Implementation Details The agent plays each game through a multi-turn ReAct-style loop, alternating between reasoning over a textual board rendering and emitting one action per turn, until the episode ends on a solve, an illegal move, or an exhausted turn budget (Appendix C.1). Sokoban, Minesweeper, and Rush Hour share this interface but differ in their state encodings and step dynamics, which we detail in Appendix C.2. For each game we build a dedicated solver that returns, for any reachable state, the optimal action value $Q^{\pi_{\text{Solver}}}$ and state value $V^{\pi_{\text{Solver}}}$: weighted A* search for Sokoban, constraint-satisfaction solving with exact combinatorial mine-probability inference for Minesweeper, and a precomputed full-state distance table via multi-source reverse BFS for Rush Hour (Appendix C.3). We query these solvers on the states visited during rollouts to form the solver-advantage signal.

CAST is implemented on top of the DAPO backbone, from which we retain the token-mean loss and clip-higher objective, augmenting its outcome objective with the solver-advantage process signal

Table 1: **Main results on the training games.** Avg@4 success rate (%) on in-domain (ID) and unseen-difficulty (Unseen) levels; **Average** aggregates the three games. Within the Qwen3-4B-Instruct-2507 block, bold and underlined values mark the best and second-best results, respectively.

Type	Approach	Sokoban		Minesweeper		Rush Hour		Average	
		ID	Unseen	ID	Unseen	ID	Unseen	ID	Unseen
Closed-Source Models (ReAct; G = Gemini)									
Prompting	G-2.5-Flash	64.0	33.3	38.1	23.0	73.9	63.6	58.7	40.0
Prompting	G-2.5-Pro	77.5	45.6	50.8	18.8	92.0	88.9	73.4	51.1
Prompting	Sonnet-4.5	60.0	29.3	34.4	22.8	56.9	42.3	50.4	31.5
Prompting	Opus-4.5	71.9	37.3	47.3	29.4	70.4	59.1	63.2	41.9
Prompting	Sonnet-4.6	79.5	47.1	57.6	42.5	88.1	81.4	75.1	57.0
Prompting	Opus-4.6	82.5	51.5	64.6	55.3	92.6	85.3	79.9	64.0
Base Model: Qwen3-4B-Instruct-2507									
Prompting	ReAct	44.3	16.8	4.5	0.4	1.1	0.5	16.6	5.9
RL Training	GRPO	71.3	30.5	9.7	0.5	<u>53.8</u>	26.7	44.9	19.2
RL Training	GSPO	<u>72.6</u>	28.3	28.2	5.0	24.8	11.5	41.9	14.9
RL Training	DAPO	71.9	<u>31.3</u>	<u>29.8</u>	<u>7.3</u>	32.5	17.4	44.7	18.7
RL Training	GiGPO	70.6	29.6	13.7	1.4	51.9	<u>31.3</u>	<u>45.4</u>	<u>20.8</u>
RL Training	CAST (Ours)	**77.0**	**34.8**	**44.7**	**11.0**	**64.7**	**39.5**	**62.1**	**28.4**

described in Section 2. Following common practice, all environments use a 0/1 sparse reward. We train with a batch size of 16, 8 rollouts per prompt, and a maximum response length of 16384 tokens; the solver-advantage weight is set to α=0.1 by default, and the remaining training settings are listed in Appendix C.6. For evaluation, we score every game by success rate. To reduce sampling variance, for each instance we draw four independent rollouts and report the average success rate, denoted Avg@4. Unless otherwise noted, all tables report the mean over **3 independent runs**, taking each run's final checkpoint at step 200 (Sokoban), 400 (Minesweeper), and 300 (Rush Hour) for both within-game and OOD-transfer evaluation; closed-source references use a single run under the same Avg@4 protocol.

3.2 Main Results on the Training Games

Comparison with training-free baselines. Training substantially improves the *Qwen3-4B-Instruct-2507* base policy. With a ReAct framework, the frozen base reaches game-averaged Avg@4 scores of 16.6 on ID and 5.9 on Unseen-Difficulty; after training, our 4B agent reaches 62.1 and 28.4. On ID, it also exceeds the ReAct scores of two closed-source models, Gemini-2.5-Flash (58.7) and Claude-Sonnet-4.5 (50.4), showing the practical benefit of solver-guided RL beyond prompting alone for this base policy.

Controlled comparison against trained baselines. Trained from the same base with the same final reward, CAST attains the best Avg@4 in every game under both evaluation settings. The cleanest comparison is against DAPO, our backbone with the solver-advantage signal removed: adding this signal lifts the ID average from 44.7 to 62.1 and the Unseen-Difficulty average from 18.7 to 28.4, isolating its contribution. The lead also holds over the outcome-only baseline GRPO (44.9 ID, 19.2 Unseen) and the process-level baseline GiGPO (45.4, 20.8), and spans all three games rather than coming from a single environment. Minesweeper shows the largest ID gain of $+14.9$ over the best trained baseline, while its Unseen-Difficulty score of 11.0 leaves room for improvement on harder instances.

Training dynamics. Figure 3 shows that solver guidance improves both training efficiency and validation performance. CAST attains the highest validation Avg@4 on all three games and reaches DAPO's peak after only 120, 200, and 140 training steps on Sokoban, Minesweeper, and Rush Hour, respectively, compared with 200, 400, and 240 steps for DAPO, corresponding to a 1.7–2.0$\times$ speedup. CAST leads on training reward across all three games, clearly on the harder Minesweeper and Rush Hour and edging ahead only late on the easier Sokoban.

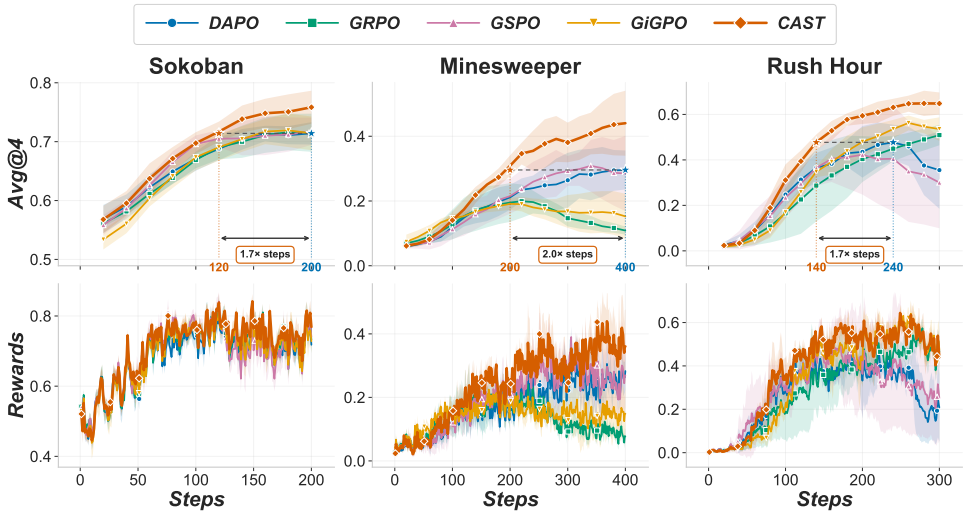


Figure 3: **Training dynamics.** Horizontal dashed lines mark DAPO's peak validation Avg@4; vertical dotted lines mark when CAST and DAPO first reach it, in orange and blue respectively. Curves are EMA-smoothed at 0.6, with bands showing the standard deviation over three runs.

Table 2: **Zero-shot OOD transfer.** Avg@4 success rate (%) on ALFWorld and WebShop; **Average** aggregates source-game agents, and **Overall** aggregates both domains. Dashes denote entries not applicable to ReAct; bold and underlined values mark the best and second-best results, respectively.

Type	Method	ALFWorld				WebShop				Overall
		Sok.	*Mine.*	*Rush*	**Average**	*Sok.*	*Mine.*	*Rush*	**Average**	
Base: Qwen3-4B-Instruct-2507										
Prompting	ReAct	—	—	—	30.2	—	—	—	<u>18.8</u>	24.5
RL Training	GRPO	31.5	22.3	27.9	27.2	<u>20.4</u>	14.0	18.0	17.5	22.4
RL Training	GSPO	34.2	22.0	<u>40.0</u>	<u>32.1</u>	18.0	13.1	**20.9**	17.3	<u>24.7</u>
RL Training	DAPO	<u>37.3</u>	**35.2**	18.6	30.4	17.5	18.8	13.6	16.6	23.5
RL Training	GiGPO	35.6	24.9	32.1	30.9	17.2	<u>20.2</u>	16.4	17.9	24.4
RL Training	CAST (Ours)	**38.4**	<u>33.9</u>	**41.4**	**37.9**	**23.2**	**25.1**	<u>19.7</u>	**22.7**	**30.3**

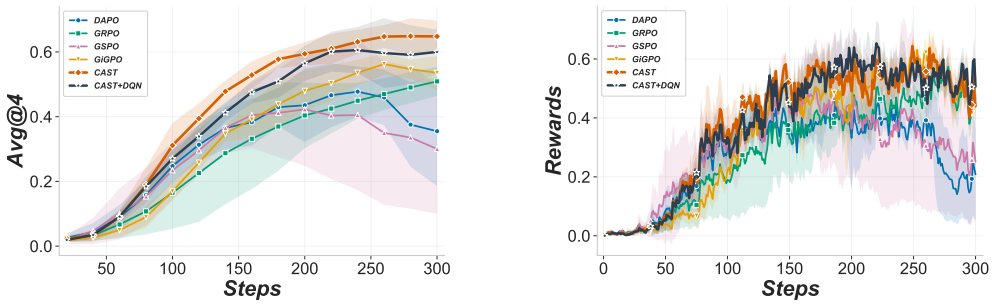
3.3 Zero-Shot OOD Transfer

Table 2 shows that the benefits of solver-guided training extend beyond the source games. CAST achieves the highest domain average on both ALFWorld (37.9) and WebShop (22.7), exceeding the strongest trained baseline by 5.8 and 4.8 points, respectively. It also outperforms training-free ReAct on both domains and attains the best Overall score of 30.3, 5.6 points above the second-best method. A WebShop case study is provided in Appendix B.3.

3.4 Ablation Studies

We ablate the key design choices of CAST on Sokoban.

1. **Solver-advantage weight** α (see Figure 6 in Appendix D, left): the weight trades off the process signal against the outcome objective, and our default α=0.1 is best on both validation Avg@4 and training reward. When α is too small the process signal is diluted and the run behaves like the outcome-only baseline, converging slowly to a lower plateau. When α is too large the signal crowds out the sparse outcome objective: validation Avg@4 climbs quickly at first but peaks early and then declines, and training reward becomes unstable and drops in late training. Only α=0.1 keeps improving and stays stable throughout.
2. **Transformation and normalization** (see Figure 6 in Appendix D, right): the `asinh` transformation and batch-level RMS normalization play complementary roles, and removing either



(a) Val Avg@4 (b) Train rewards

Figure 4: **Learned value network as a solver on Rush Hour.** We replace the exact solver with a DQN-based value network trained without solver distances as the process-signal source. Curves are EMA-smoothed at 0.6; bands show the standard deviation across the 3 runs.

lowers the final Avg@4. Dropping `asinh` hurts most: without it the run is slowest and stays lowest throughout, since unbounded solver advantages let a few large values dominate the update. Dropping RMS normalization instead matches the full method early but plateaus and dips late, with noisier training reward, indicating that per-batch rescaling is what keeps the signal stable as training progresses.

We analyze the practicality of solver guidance from two angles: the training-time overhead it adds, and whether it still helps when the exact solver is replaced by an approximate learned value network.

Solver runtime overhead. Forming the solver-advantage signal requires one solver query per environment step to obtain $Q^{\pi_{\text{Solver}}}$ and $V^{\pi_{\text{Solver}}}$ for the current state. Figure 5 shows that the relative cost of these queries rapidly diminishes at broader levels of the training pipeline. Although solver queries account for 8.4% of an environment step, environment interaction itself occupies only 0.1% of trajectory runtime, with the remaining 99.9% dominated by LLM generation. Solver time therefore accounts for only 0.01% of a trajectory. Since rollout collection occupies 60.7% of a training step, the solver's estimated end-to-end contribution is just 73 ppm of total training-step wall-clock time, making its overhead negligible relative to generation and policy optimization.

Learned value networks as solvers. Solver guidance can also use an approximate learned value function as the process-signal source, which extends it to the traditional RL or deep RL. On Rush Hour, we replace the exact solver with a DQN-based value network (Van Hasselt et al., 2016) trained without solver distances, then refined by self-distillation and a small geometric prior; we use its state values as the process signal (details in Appendix C.4). As shown in Figure 4, the learned-network variant closely tracks the exact-solver version and ends only slightly below it in validation Avg@4. It remains above the trained baselines at the end of training and, unlike DAPO and GSPO, shows no comparable late-stage drop in validation Avg@4. Thus, even with a learned value function, CAST retains much of the benefit of exact-solver guidance, showing that solver guidance carries over to standard RL rather than depending on a hand-built solver.

4 Conclusion

We address turn-level credit assignment in RLVR for LLM game agents under sparse terminal rewards. CAST queries a game solver on LLM-visited states and converts per-action changes in the solver's state value into solver advantages that provide fine-grained, turn-level credit; under a soft-optimal solver assumption, this update also admits a logit-free on-policy distillation interpretation without teacher logits. Across Sokoban, Minesweeper, and Rush Hour, CAST achieves the best performance among trained methods on every game under both in-domain and unseen-difficulty evaluation, reaches DAPO's peak validation performance in 1.7–$2.0\times$ fewer training steps, and attains the highest average zero-shot performance on held-out ALFWorld and WebShop. Solver queries add negligible training overhead, while an approximate learned value network retains much of the benefit of exact solver guidance.

AI USE STATEMENT

We used generative AI tools to assist in writing up the proofs in Appendix C.5, to give feedback on the experimental methodology, and to implement the methods, including the CAST objective, the three game solvers (Appendix C.3), and the learned value network (Appendix C.4). The conceptual framework behind CAST and the theoretical results themselves were developed by the authors. We did not use generative AI to generate synthetic datasets, clean or reformat data, assist with translation, support qualitative or thematic analysis: all game instances come from our own procedural generators (Appendix B.1), and every conclusion drawn from the results was reached by the authors.

We additionally used generative AI tools to draft and edit parts of this paper for readability, to write and edit code, to produce visualizations of experimental data, and to search for and summarize related literature. The research question and the gap it addresses were identified by the authors, and every citation was selected, added, and formatted by the authors.

We have reviewed all AI-assisted work: every proof was checked by the authors against their own derivation; all AI-assisted code was tested by the authors, and the reported results were reproduced from that code; every citation was checked against the original publication to confirm that the work exists and that the claim attributed to it is accurate. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects, crowdsourcing, or personal data. All training and evaluation instances for Sokoban, Minesweeper, and Rush Hour are procedurally generated by our own generators (Appendix B.1), and ALFWorld and WebShop are public benchmarks used under their original licenses in simulated environments (Appendix B.2); closed-source models are accessed only through their official APIs. Our method improves the decision-making of LLM agents in sandboxed games and simulated text environments; as with any agent-training technique, stronger agents deployed in real web or embodied settings would call for appropriate safeguards and human oversight. We are not aware of any conflicts of interest or other ethical concerns raised by this work.

REPRODUCIBILITY STATEMENT

We describe the CAST objective in Section 2 and state all assumptions and complete proofs of the theoretical results in Appendix C.5. Data generation, difficulty tiers, and train/test de-duplication are detailed in Appendix B.1, and the OOD benchmark protocols in Appendix B.2. The agent loop and verbatim system prompts are given in Appendix C.1, the environment and solver implementations in Appendix C.2 and Appendix C.3, and the learned value network in Appendix C.4. All training hyperparameters and evaluation settings for every method are listed in Appendix C.6, and all reported results are averaged over 3 independent runs (Section 3). We will release the code, game generators, solvers, and evaluation scripts upon publication.

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.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

Anthropic. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5, November 2025a.

Anthropic. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, September 2025b.

Anthropic. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026a.

Anthropic. Introducing claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, February 2026b.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Jiaze Chen, Xuefeng Li, Qiying Yu, et al. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. *Advances in Neural Information Processing Systems*, 38:3613–3661, 2026a.

Liang Chen, Hongcheng Gao, Tianyu Liu, Zhiqi Huang, Flood Sung, Xinyu Zhou, Yuxin Wu, and Baobao Chang. G1: Bootstrapping perception and reasoning abilities of vision-language model via reinforcement learning. *arXiv preprint arXiv:2505.13426*, 2025.

Yuxin Chen, Yu Wang, Yi Zhang, Ziang Ye, Zhengzhou Cai, Yaorui Shi, Qi Gu, Hui Su, Xunliang Cai, Xiang Wang, et al. Learning to self-verify makes language models better reasoners. *arXiv preprint arXiv:2602.07594*, 2026b.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35: 18343–18362, 2022.

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

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena. *arXiv preprint arXiv:2504.11442*, 2025.

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

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *Advances in Neural Information Processing Systems*, 38:114399–114431, 2026.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

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

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are llms at playing games? *arXiv preprint arXiv:2505.15146*, 2025.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1-2):219–251, 2001.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning*, pp. 267–274, 2002.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron C. Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in RL training of llms. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL `https://proceedings.mlr.press/v267/kazemnejad25a.html`.

Junbo Li, Peng Zhou, Rui Meng, Meet P Vadera, Lihong Li, and Yang Li. Turn-ppo: Turn-level advantage estimation with ppo for improved multi-turn rl in agentic llms. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6227–6243, 2026.

Yi-Chen Li, Tian Xu, Yang Yu, Xuqin Zhang, Xiong-Hui Chen, Zhongxiang Ling, Ningjing Chao, Lei Yuan, and Zhi-Hua Zhou. Generalist reward models: Found inside large language models. *arXiv preprint arXiv:2506.23235*, 2025.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. Gameworld: Towards standardized and verifiable evaluation of multimodal game agents. *arXiv preprint arXiv:2604.07429*, 2026.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. In *International Conference on Learning Representations*, volume 2025, pp. 96666–96702, 2025.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

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

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. In *International Conference on Machine Learning*, pp. 58658–58725. PMLR, 2025.

Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chen Gao, Chen Zhang, Chengcheng Han, et al. Longcat-flash-thinking-2601 technical report. *arXiv preprint arXiv:2601.16725*, 2026.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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*, 2025.

Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, Qineng Wang, Hanyang Chen, Yiping Lu, Zhengyuan Yang, Lijuan Wang, et al. Vagen: Reinforcing world model reasoning for multi-turn vlm agents. *Advances in Neural Information Processing Systems*, 38:172871–172933, 2026.

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.

Yunfei Xie, Yinsong Ma, Shiyi Lan, Alan Yuille, Junfei Xiao, and Chen Wei. Play to generalize: Learning to reason through game play. *arXiv preprint arXiv:2506.08011*, 2025.

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.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Ziang Ye, Wentao Shi, Yuxin Liu, Yu Wang, Zhengzhou Cai, Yaorui Shi, Qi Gu, Xunliang Cai, and Fuli Feng. Look before you leap: Autonomous exploration for llm agents. *arXiv preprint arXiv:2605.16143*, 2026.

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

Yangbin Yu, Mingyu Yang, Junyou Li, Yiming Gao, Feiyu Liu, Yijun Yang, Zichuan Lin, Jiafei Lyu, Yicheng Liu, Zhicong Lu, et al. Proact: Agentic lookahead in interactive environments. *arXiv preprint arXiv:2602.05327*, 2026b.

Yi-Kai Zhang, Zhiyuan Yao, Hongyan Hao, Yueqing Sun, Qi Gu, Hui Su, Xunliang Cai, De-Chuan Zhan, and Han-Jia Ye. v_0: A generalist value model for any policy at state zero. *arXiv preprint arXiv:2602.03584*, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In Dieter Fox and Carla P. Gomes (eds.), *AAAI*, pp. 1433–1438. AAAI Press, 2008.

APPENDIX

A RELATED WORK

LLMs as Game Agents. Recent work has moved foundation models from passive text generation toward interactive game play. Prompted and scaffolded agents equip LLMs or VLMs with memory, planning, reflection, and coordination, showing that they can interact with games in open world exploration (Wang et al., 2023), general computer and video game control (Tan et al., 2025), and reflective or multi-agent settings (Shinn et al., 2023), often without updating model weights. In parallel, game environments and benchmarks make such interaction measurable and scalable: open world platforms (Fan et al., 2022) and recent benchmarks for LLMs and VLMs (Paglieri et al., 2025; Hu et al., 2025; Guertler et al., 2025; Ouyang et al., 2026) provide testbeds for evaluating and training game agents. Game RL with outcome supervision then turns final rewards from games and puzzles into training signals for improving reasoning, transfer, or performance in specific games (Xie et al., 2025; Chen et al., 2025; 2026a; Liu et al., 2025). These signals are clean and verifiable, but remain coarse for long trajectories: they indicate whether an attempt succeeds, while providing limited guidance about which intermediate decisions mattered.

Credit Assignment and Process Signals in Agentic RL. To address the coarse credit provided by final outcomes, recent agentic RL methods construct process signals below the trajectory level, at the segment, turn, or token (Li et al., 2026; Guo et al., 2026; Wang et al., 2025). Some methods redistribute outcome information through grouping or decomposition, combining trajectory-level advantages with step-level comparisons over recurring or comparable states (Feng et al., 2026; He et al., 2026). Others learn process evaluators for agent trajectories, including process reward models that estimate stepwise promise and progress (Xi et al., 2026), and learned turn-level critics for multi-turn environments (Wang et al., 2026). Search- and rollout-based approaches instead estimate intermediate values by expanding possible continuations, producing value estimates or process labels for policy improvement (Hao et al., 2023; Kazemnejad et al., 2025; Yu et al., 2026b). These signal sources expose a recurring trade-off: grouping methods require comparable intermediate states, learned verifiers and critics require additional training signals or models, and search or rollout estimates can add substantial computation or depend on the rollout policy. In contrast, our work uses game-specific solvers as an external source of turn-level supervision.

Logit-Free Distillation from Solvers and Search. CAST is also related to distillation, but differs in the teacher signal it uses. Recent LLM on-policy distillation learns from student-generated outputs, but assumes access to teacher token distributions or log probabilities (Agarwal et al., 2024; Gu et al., 2024; Lu & Lab, 2025). Classical solvers instead provide structured signals such as actions, feasibility, and cost-to-go values, connecting our setting to traditional imitation learning and search distillation. DAgger queries an oracle on learner-visited states (Ross et al., 2011), AggreVaTe supervises actions with cost-to-go estimates (Ross & Bagnell, 2014), and expert-iteration methods distill search-improved decisions into a policy (Anthony et al., 2017; Silver et al., 2017). These methods target task-specific policies, whereas we attach scalar solver advantage estimates to LLM-sampled actions, yielding a logit-free process signal that plugs directly into RLVR.

B DATASETS AND BENCHMARKS

B.1 DATASET DETAILS

Data generation. Each game uses a procedural generator that produces instances at a controllable difficulty. **Sokoban** rooms are grown by random wall placement and then populated by a reverse-playing search, which guarantees a solvable layout in which at least one box must be pushed. **Minesweeper** boards place mines uniformly at random outside a 3×3 safe zone around a designated first-click cell, which the environment reveals at the start of every episode so that the board is fixed from the agent's first action onward; we do not filter for no-guess solvability, so a few boards may still require a probabilistic guess, which the oracle handles as described in Appendix C.3. **Rush Hour** boards are sampled at random and kept only if an exact solver confirms an optimal solution whose length lies in a target range; this solver is used only for data construction and differs from the training oracle of Appendix C.3.

Difficulty and splits. Difficulty is controlled by board size and a game-specific count: boxes for Sokoban, mines for Minesweeper, and total vehicles for Rush Hour. For each game, we train on a single in-domain (ID) tier and hold out a harder unseen-difficulty tier, obtained by increasing this count and, for Sokoban and Minesweeper, enlarging the board; both tiers are listed in Table 3. Each tier provides 200 evaluation instances. All instances are drawn from independent random seeds and de-duplicated by a hash of their initial board, so that no instance recurs within the training set and no evaluation instance appears in training. The textual state encoding and action interface of each environment are described in Appendix C.2.

Table 3: Per-game in-domain (ID) and unseen-difficulty (Unseen) tiers. Agents are trained only on ID and evaluated on both tiers; for Rush Hour, the count includes the target vehicle `A`.

Game	Split	Board	# Boxes, Mines, or Vehicles
Sokoban	ID	6×6	2 boxes
	Unseen	7×7	3 boxes
Minesweeper	ID	6×6	7 mines
	Unseen	7×7	10 mines
Rush Hour	ID	6×6	7 vehicles
	Unseen	6×6	9 vehicles

B.2 OOD Transfer Benchmark Details

To measure cross-domain transfer, we take each agent trained on the three games and evaluate it zero-shot, with no further training, on two standard text-agent benchmarks, ALFWorld and WebShop. Both are driven by the same multi-turn loop and single-action interface as the games (Appendix C.1), differing only in their observations and action space, so that any success reflects abilities transferred from game training rather than benchmark-specific tuning.

ALFWorld. ALFWorld (Shridhar et al., 2021) is an embodied household benchmark in which the agent completes a natural-language instruction (e.g. heating an object and placing it) by navigating and manipulating objects in a text-rendered room. We use its TextWorld engine over the six standard task types and evaluate on the *seen* validation split. Each turn presents the current observation together with the list of admissible commands, and the agent replies with one command such as `go to`, `take`, `open`, or `heat`; an invalid command is rejected with feedback and the episode continues. An episode runs for up to 30 turns.

WebShop. WebShop (Yao et al., 2022) is a simulated online store in which the agent must fulfill a shopping instruction by searching, browsing, and purchasing a product that matches the requested attributes and price. The text observation renders the current page, and the agent acts through `search[query]` and `click[button]`, choosing from the clickable elements listed for that page. We evaluate on the standard 500 test instructions, with each episode capped at 30 turns. WebShop reports two metrics, a continuous match score and a binary success rate; we use the success rate, counting an instruction as solved only on a perfect match.

Both benchmarks are decoded with the same open-source configuration as the games (Table 8): SGLang serving, temperature 0.6, top-p 0.95, up to 16,384 tokens per turn and 2,048 prompt tokens, and 4 rollouts per instance.

B.3 OOD Transfer Case Study

We compare the base model *Qwen3-4B-Instruct-2507* and its Sokoban-trained model on the same WebShop instruction; the latter receives only solver-guided Sokoban training before zero-shot evaluation. The instruction asks, "*Find me easy clean sofa tables with solid wood for living room with color: walnut, and price lower than 260.00 dollars*". WebShop credits a purchase only on a full attribute match; here, the `walnut` option must be selected on the product page before the irreversible `buy now`. The base model commits to an invalid brown table, whereas the CAST-trained

model rejects a poor candidate, searches again, and selects the required color before buying. This contrast illustrates recovery from an intermediate mistake and constraint checking before commitment, behaviors consistent with the planning discipline learned in Sokoban. Every action is retained, while some reasoning is omitted for brevity and marked as "[. . .]".

Base model (untrained) — **3 steps – reward 0.0**

S0 obs: search page.
think: "A good search term would be 'solid wood sofa table' as it captures the essential product type and material. This avoids over-specificity and allows the system to return relevant results that can be filtered later for color and price."

```
search[solid wood sofa table]
```

S1 obs: 50 results; none of the listed titles says *walnut*.
think: "[. . .] The most suitable product that meets the solid wood and price criteria is the 'Wyndenhall' table, even though it's brown. It is solid wood, priced under $260, and fits the purpose."

```
click[b084hbqytd]
```

S2 ★ obs: Wyndenhall product page ("Natural Aged Brown"); it exposes *no* color option.
think: "[. . .] We cannot do a second one with 'walnut' because that would include color — which violates the rule. Therefore, we cannot search again. [. . .] we should proceed with buying the best available match."
→ *over-applies a prompt tip, refuses to explore, and commits the irreversible* `buy` *on a page that can never satisfy the goal.*

```
click[buy now]wrong item - reward 0.0
```

CAST (Sokoban-trained, zero-shot) — **6 steps – reward 1.0**

S0 obs: search page.
think: "Since I cannot include size or color in the search term (as per tips), I should search for 'sofa tables solid wood' to get relevant products. [. . .]"

```
search[sofa tables solid wood]
```

S1 obs: 50 results; none of the titles says *walnut*.
think: "The best match in terms of solid wood and low price is the 'SunnyPoint Classic Side Table' (B084ZPN17C), even though it's not walnut. [. . .] I will click on the SunnyPoint table, assuming it might be a close enough match for the task."

```
click[b084zpn17c]
```

S2 ★ obs: SunnyPoint side table page — black, metal frame, not solid walnut wood.
think: "The current product does not meet the color requirement. I should not proceed with this one. I need to go back to search with a refined term that includes 'walnut' to get better results. [. . .] I will now search for 'sofa tables walnut solid wood' to target the exact color and material."
→ *rejects a bad candidate and spends its second search to recover, rather than settling.*

```
search[sofa tables walnut solid wood]
```

S3 obs: re-ranked results; the Baja solid-wood table ($237) now appears near the top.
think: "The Baja Night Stand (B079N3VLRJ) is solid wood, priced under $260, and is suitable for living room use. [. . .] I will proceed to click on it as the best available option within constraints."

```
click[b079n3vlrj]
```

S4 ★ obs: Baja product page exposes a color selector: `brown / grey / walnut / white`.
think: "The available color options in the page show: brown, grey, walnut, white. I should click on the 'walnut' color option to see if any product with walnut finish appears. [. . .] This is the best next step to fulfill the color requirement."
→ *sees a* `walnut` *option and selects it* before *buying — the move that actually registers the required attribute.*

```
click[walnut]
```

S5 obs: page looks unchanged after selecting `walnut`.
think: "Clicking on 'walnut' did not change the product listing [...] I should buy this product as the best available option. Therefore, I will proceed with 'Buy Now' for this product."
`click[buy now]` **`perfect match - reward 1.0`**

C Implementation Details and Hyperparameters

C.1 Agent Implementation

All three games share a single agent scaffold, with game-specific system prompts, observation formatting, action parsers, and feedback. At each turn the agent appends the environment's text observation of the current board to the context and asks the policy to reason and then commit to one action, which it parses and passes back to the environment; the loop repeats until the game is solved, fails, or reaches its turn budget.

Observation and action format. Each turn shows the current board together with the number of remaining steps, and, when the previous action was invalid or ineffective, a short line of corrective feedback. Sokoban and Rush Hour render the board as a symbolic grid paired with a coordinate listing of each entity, while Minesweeper shows the symbolic grid with row and column index headers but no separate coordinate listing. The policy is asked to reason and then place a single action inside a fenced block, such as ```` ```Up``` ```` for Sokoban, ```` ```reveal 3 2``` ```` for Minesweeper, or ```` ```A+2``` ```` for Rush Hour. We read the last fenced block in the response, so intermediate reasoning is ignored, and parse it with a per-game rule into a concrete action.

History and trajectories. The agent conditions on the full context history, including its own reasoning, so that turn t sees every earlier observation and response. The resulting trajectory, the ordered sequence of observations, responses, actions, and the terminal $0/1$ reward, is the token-level rollout optimized by the RL objectives in Appendix C.6.1.

Invalid actions and termination. An unparseable or illegal action, such as an unknown token, an out-of-bounds cell, or a move blocked by a wall, does not end the episode: it leaves the state unchanged, consumes one turn, and returns corrective feedback so the policy can retry. An episode ends only when the game is solved, a game-specific failure occurs (a mine hit in Minesweeper, or a deadlock in Sokoban), or the turn budget is exhausted, and yields reward 1 only on success.

System prompts. Each environment uses a fixed system prompt that specifies its rules and action format. For reproducibility, we list the exact prompt of every environment below: the three training games first, followed by the two OOD transfer benchmarks.

Sokoban

````
You are a player solving a Sokoban puzzle.

Sokoban Quick Guide
Goal: Push all boxes (X) onto targets (O).

Symbols:
# Wall | _ Empty | O Target | X Box | * Box on target | P Player | S Player on target

Rules:
1. You can push a box by moving into it if the space behind it is free.
2. You cannot pull boxes.
3. Walls are impassable.

Valid Action (separated by | ):
Up | Down | Left | Right

You will be provided the current observation, please decide on the next action.
You should show your thought process and then input the final action in a ``` block.
You should only output the NEXT ACTION at each iteration in the ``` block.
For example, if you want to move up, you should output: ```Up```
````

```
Please show your thinking process and put the final action in a ``` block.
In every turn, the final action MUST be one of Up, Down, Left, Right.
```

Minesweeper

```
You are a player playing a Minesweeper game.

Minesweeper Rules:
- The board is a grid of cells. Some cells contain hidden mines.
- Your goal is to reveal all cells that do NOT contain mines.
- When you reveal a cell:
  - If it contains a mine, the game is over (you lose).
  - If it does not contain a mine, a number is shown indicating how many of its 8 neighboring cells
contain mines (0-8).
  - If the number is 0, all adjacent cells are automatically revealed.
- You may also place or remove a flag on a cell to mark or unmark it as a suspected mine.
- Removing flags is allowed and encouraged if your later reasoning shows the flag was incorrect.

Board Symbols:
  .  Unrevealed cell
  F  Flagged cell (suspected mine)
 0-8 Revealed cell showing the count of adjacent mines
  *  Mine (shown only when you hit one)

Actions (one per turn):
  reveal R C  -- Reveal the cell at row R, column C.
  flag R C    -- Toggle a flag on the cell at row R, column C (place a flag if none exists, or REMOVE
 it if already flagged).

You should show your thinking process, then output exactly ONE action inside a ``` block.
For example: ```reveal 3 2``` or ```flag 0 4```
Use logic and deduction to avoid mines. Think step-by-step before acting.
```

Rush Hour

```
You are a player solving a Rush Hour sliding-block puzzle.

Rush Hour Quick Guide
Goal: Slide the red car (labeled A) horizontally until it reaches the exit on the right edge of its
row.

Board:
- A 6x6 grid of cells, zero-indexed (row, col).
- Cars and trucks are labeled A, B, C, ... Each occupies 2 or 3 cells in a line.
- The red car A is horizontal and sits on the exit row; the exit is on the right edge of that row.
- Horizontal cars can only move left or right. Vertical cars can only move up or down.
- Cells marked 'x' are walls and never move. Empty cells are '.' (or 'o').
- Cars cannot overlap, pass through each other, or pass through walls.

Action format:
- Output a single move as <car><sign><steps>.
- For a horizontal car: '+' moves right, '-' moves left; For a vertical car: '+' moves down, '-'
moves up.
- <steps> is the number of cells to slide (omit it to slide 1 cell).
- All-or-nothing: <steps> must be exactly how far the car can slide. Overshooting (more than the free
 cells) does nothing.
- One slide of any distance counts as ONE move; prefer a single multi-cell slide (e.g. A+3) over
several 1-cell moves.
- Examples: A+2 (move car A right 2 cells), B-1 (move car B up 1 cell), C+3 (move car C down 3 cells).


You will be provided the current observation, please decide on the next action.
You should only output the NEXT ACTION at each iteration in the ``` block.
For example, if you want to move car A right 2 cells, you should output: ```A+2```
Please put the final action in a ``` block.
```

ALFWorld

```
You are an expert agent completing household tasks in an embodied environment. Your task is to
complete household tasks by navigating rooms and interacting with objects.

Action Space:
Your action must be one of the following, strictly following the command (argument) format.
```

```
Navigation & Observation
- look: Look around your current location to get more details.
- inventory: Check the object you are currently holding (you can only hold one).
- go to (receptacle): Move to a receptacle (e.g., table, fridge, sink).

Interacting with Receptacles
- open (receptacle): Open a receptacle.
- close (receptacle): Close a receptacle.

Interacting with Objects
- take (object) from (receptacle): Pick up an object from a receptacle.
- move (object) to (receptacle): Place the object you are holding into or onto a receptacle.
- examine (object): Examine an object closely to learn its properties.

Changing Object States
- heat (object) with (receptacle): Heat an object with a device (e.g., microwave).
- cool (object) with (receptacle): Cool an object with a device (e.g., fridge).
- clean (object) with (receptacle): Clean an object with a device (e.g., sink).
- slice (object) with (object): Slice an object using a sharp object (e.g., knife).

Important Rules:
1. You must first navigate to a location before interacting with objects there
2. You can only hold one object at a time
3. Some objects need to be heated, cooled, or cleaned before placing them
4. Always check available actions to see what actions are currently valid

Response Format:
Think through your approach step by step, then provide your action in a ``` block.
For example: ```go to countertop 1```
Always provide exactly ONE action at a time.
```

WebShop

```
You are an expert agent completing a web-based shopping task. You need to interact with the web
environment to complete the task.

Available actions:
- search[query]: Search for products using keywords
- click[element]: Click on a page element (product link, option, Buy Now, etc.)

You will be provided the current state of the webpage, please decide on the next action.
You should show your thought process and then put the final action in a ``` block.
You should only output the NEXT ACTION at each iteration in the ``` block.
For example, if you want to search for shoes, you should output: ```search[shoes]```
If you want to click Buy Now, you should output: ```click[Buy Now]```
Please show your thinking process and put the final action in a ``` block.
Output exactly ONE action per turn.
```

C.2 Environment Implementation

The three environments share the same interface but differ in their state encoding, action semantics, and termination rules. Sokoban and Rush Hour pair a symbolic grid with an explicit zero-indexed coordinate listing, whereas Minesweeper uses a symbolic grid with indexed rows and columns. All environments give a single sparse terminal reward of 1 on success and 0 otherwise.

Sokoban. The board uses the symbols `#` (wall), `_` (empty), `O` (target), `X` (box), $\star$ (box on target), `P` (player), and `S` (player on target). The four actions `Up/Down/Left/Right` move the player, pushing a box one cell when the cell behind it is free; boxes cannot be pulled and walls are impassable. The episode ends in success when every box covers a target, or otherwise on a detected deadlock or the step budget.

Minesweeper. Only revealed information is shown: an unrevealed cell is `.`, a flag is `F`, a revealed count is a digit `0`–`8`, and a mine is $\star$ (shown only when hit); a numeric header indexes the rows and columns. The actions are `reveal R C` and `flag R C`, where a flag toggles a marker and revealing a 0 cell flood-fills its zero-valued neighborhood. The episode ends in failure on hitting a mine, in success once every non-mine cell is revealed, or on the step budget. Each instance fixes a latent mine layout and a first-click cell that the environment reveals before the agent acts, so every rollout starts from the same partially revealed board; this removes the initial blind guess that standard Minesweeper would otherwise require.

Rush Hour. Vehicles are labeled A, B, C, ... by index, with the target red car A horizontal on the exit row; empty cells are ., walls are x, and the exit lies at the right edge of A's row. An action ⟨car⟩⟨±⟩⟨steps⟩ slides one vehicle along its orientation (+ is right or down, − is left or up) by a given number of cells, and is all-or-nothing: it executes only if every intermediate cell is free, so a blocked or overshooting move leaves the board unchanged. The episode ends in success when A reaches the exit, or otherwise when the step budget is exhausted.

C.3 Solver Implementation

Each game provides a solver-derived cost $N(s)$ that measures the remaining work from state s. We define the potential $\Phi(s) = -N(s)$ and per-step oracle advantage $A(s_t, a_t) = \Phi(s_{t+1}) - \Phi(s_t) = N(s_t) - N(s_{t+1})$. In the shortest-path domains, this gives $+1$ to an optimal move, 0 to a wasteful move, and a negative score to harmful moves. Minesweeper uses the analogous solver-completion cost defined below. The solver is queried on states visited along each rollout.

Sokoban. We compute $N(s)$ by A* over states $(\textit{player}, \{\textit{boxes}\})$ with a minimum-cost box-to-target assignment as an admissible heuristic. The search returns the shortest solution within its node budget and remains cheap because the training tiers have few boxes on small grids. The same solver also supports the deadlock check used for optional early termination.

Minesweeper. Partial observability makes an optimal deterministic cost-to-go ill-defined, so here $K(s)$ measures the work of a deterministic peek-free solver. Using constraint reasoning and global mine probabilities, $K(s)$ is the number of reveal actions needed to clear all safe cells, or $+\infty$ if the solver rollout fails, including when a required probabilistic reveal hits a mine. Because each instance fixes its latent mine layout and first-click reveal (Appendix C.2), $K(s)$ is deterministic for that instance while the agent remains partially observed. The oracle advantage is $K(s_t) - K(s_{t+1})$.

Rush Hour. The training oracle uses a different solver from the one that generates the data. Data generation validates each board with a bounded IDA* search (Appendix B.1). For training, we instead precompute a shortest-path table over the board's reachable component, giving exact $N(s)$ lookups within the configured state cap; larger components fall back to bounded IDA*. One slide of any distance counts as a single move.

C.4 Learned Value Network as a Solver

To test whether an exact solver is required at deployment, we replace the Rush Hour oracle with a DQN-based value network. A masked Double DQN encodes the 6×6 board with a small transformer and outputs one Q-value per action; illegal actions are masked, and $V(s) = \max_{a\in\mathcal{A}(s)} Q(s,a)$. The backend returns the cost surrogate $\widehat{N}(s) = -\text{scale} \cdot V(s)$, so the environment computes $A(s_t, a_t) = \widehat{N}(s_t) - \widehat{N}(s_{t+1}) = \text{scale}\,[V(s_{t+1}) - V(s_t)]$. Solved states use a fixed value above the non-terminal network range, ensuring that a solving move receives positive advantage.

We train the network in three stages on the same underlying ID split used for LLM training, without generating new puzzle instances. The base network learns from a terminal success reward with a per-step penalty. We then apply two warm-started potential-regression stages: the first distills a one-step lookahead potential, while the second repeats the one-step target and adds the geometric prior $-0.05b(s)$, where $b(s)$ counts occupied cells between the red car and the exit. These optimization targets use game rewards, frozen network predictions, game dynamics, and the geometric prior, but no solver-distance targets. Hyperparameters are listed in Table 4.

C.5 Equivalence to Cross-Entropy Distillation

This section formalizes the distillation interpretation of the combined training rule Equation 8. We prove that, under four explicit assumptions, the implicit objective of the solver-shaped GRPO update is a task return regularized by the cross-entropy between the student and the solver. We then decompose this cross-entropy to reveal both a KL-divergence penalty and a mode-seeking entropy reduction, derive the closed-form optimal policy, characterize the full-domain behavior

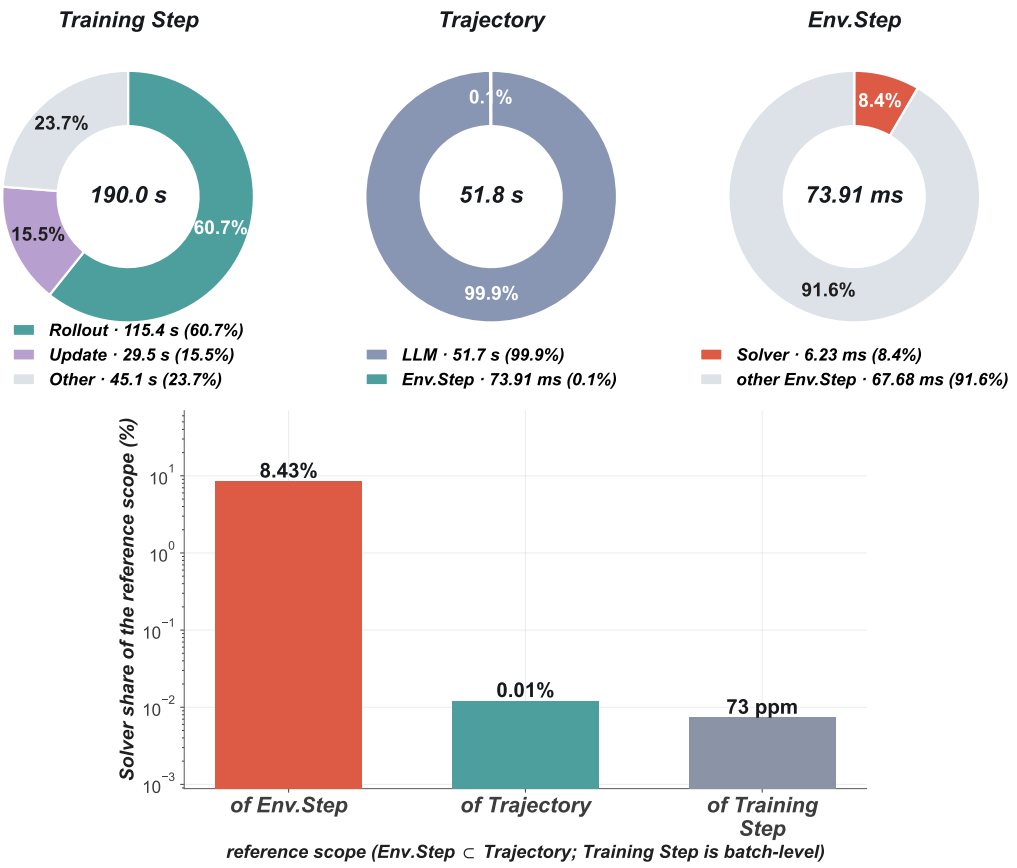


Figure 5: **Solver overhead on Sokoban. Top**: wall-clock breakdowns at three granularities. The left pie decomposes an average batch-level optimizer step (190.0 s), which includes the parallel collection of 128 trajectories (16 prompts $\times$ 8 rollouts), the policy update, and other operations. Rollout collection, policy update, and other operations account for 60.7%, 15.5%, and 23.7% of step time, respectively; the last category consists mainly of periodic validation and checkpointing. The middle pie shows that LLM generation accounts for 99.9% of the runtime of a single trajectory, while the right pie shows that solver queries account for 8.4% of a single environment step. **Bottom**: the solver's time share across increasingly broad scopes on a logarithmic scale. Accounting for parallel rollout collection, its estimated share decreases from 8.43% of an environment step to 0.01% of a trajectory and 73 ppm of a training step.

when the small-signal assumption is relaxed, and give a performance-improvement guarantee via the Performance Difference Lemma.

C.5.1 Notation

We work in the finite-horizon MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, H)$ from Subsection 2.1 with undiscounted returns ($\gamma = 1$). Let π_θ denote the student (LLM) policy and π_{Solver} the solver's implicit policy. We write $V^\pi_{\text{task}}, Q^\pi_{\text{task}}, A^\pi_{\text{task}}$ for the value, action-value, and advantage functions under the task reward, and $d^{\pi_\theta}(s) = \sum_{t=0}^{H-1} \Pr(s_t = s \mid \pi_\theta)$ for the (unnormalized) state occupancy measure of π_θ, so that $\sum_s d^{\pi_\theta}(s) = H$. We define the following standard information-theoretic quantities for any state s:

$$\mathcal{H}\big(\pi_\theta(\cdot|s)\big) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s), \tag{10}$$

$$\mathrm{KL}\big(\pi_\theta(\cdot|s) \,\|\, \pi_{\text{Solver}}(\cdot|s)\big) = \sum_a \pi_\theta(a|s) \log \tfrac{\pi_\theta(a|s)}{\pi_{\text{Solver}}(a|s)}, \tag{11}$$

$$\mathrm{H}\big(\pi_\theta(\cdot|s),\, \pi_{\text{Solver}}(\cdot|s)\big) = -\sum_a \pi_\theta(a|s) \log \pi_{\text{Solver}}(a|s). \tag{12}$$

Table 4: Training and deployment configuration of the DQN-based value network. All stages use the same architecture and warm-start from the preceding checkpoint.

Setting	Value
Encoder	Transformer, d_{model}=128, 8 heads, 3 layers, 512-d output
Base RL algorithm	Masked Double DQN, n-step 3, γ=0.99
Base reward	Terminal success indicator minus 0.01 per step
Base training steps	3×10^6
Distillation objective	Two stages of one-step potential regression
Geometric prior blend	$-0.05\times$ occupied path-cell count (second stage)
Distillation epochs	150 per stage
Learning rate	10^{-4} (base) / 10^{-3}, 5×10^{-4} (distill)
Training boards	8,000 puzzles from the ID training split
Deployment	scale=2, solved-state value =1.0

The last quantity is the cross-entropy from π_θ to π_{Solver}. These three are related by the identity:

$$\mathrm{H}(\pi_\theta,\, \pi_{\text{Solver}}) = \mathrm{KL}(\pi_\theta \,\|\, \pi_{\text{Solver}}) + \mathcal{H}(\pi_\theta), \tag{13}$$

which expresses the cross-entropy as the sum of the KL divergence and the entropy of π_θ.

Throughout, we use the *baseline invariance* property of policy gradients (Sutton et al., 1999): for any function $c(s)$ independent of the action, $\mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\cdot c(s)] = 0$.

C.5.2 Assumptions

Assumption C.1 (Soft-Optimal Solver)**.** There exists a temperature $\tau > 0$ such that the solver acts as the optimal policy of a soft (maximum-entropy) MDP (Ziebart et al., 2008; Haarnoja et al., 2018):

$$\pi_{\text{Solver}}(a|s) = \exp\Big(\tfrac{1}{\tau}\big(Q^{\pi_{\text{Solver}}}(s,a) - V^{\pi_{\text{Solver}}}(s)\big)\Big), \qquad V^{\pi_{\text{Solver}}}(s) = \tau \log \textstyle\sum_{a'} \exp\big(Q^{\pi_{\text{Solver}}}(s,a')/\tau\big). \tag{14}$$

Assumption C.2 (Small-Signal Regime)**.** The solver advantages encountered during training satisfy $|A^{\pi_{\text{Solver}}}| \lesssim 1$. Since most moves change the cost-to-go by 0 or ±1, this holds for the vast majority of training steps. Under this condition, $\operatorname{asinh}(A^{\pi_{\text{Solver}}} + 1)$ is well-approximated by its first-order Taylor expansion around 1 (the value at an optimal move). This assumption is relaxed in Corollary C.10.

Assumption C.3 (GRPO Consistency)**.** The group-relative advantage $\hat{A}_i^{\text{outcome}}$ (2) is an unbiased estimator of the task advantage $A^{\pi_\theta}_{\text{task}}$ up to a positive scaling constant (absorbed into the learning rate).

Assumption C.4 (Frozen Visitation Surrogate)**.** Following standard practice in PPO-style methods (Schulman et al., 2017), the shaping term $\alpha\, h(\widetilde{A}^{\pi_{\text{Solver}}}_{i,t})$ is treated as a per-step immediate weight, and its policy-gradient contribution is computed under a surrogate objective that holds the state visitation distribution d^{π_θ} fixed at the current iterate.

C.5.3 Lemmas

Lemma C.5 (Advantage–Log-Probability Identity)**.** *Under Assumption C.1, for all* (s, a)*:*

$$A^{\pi_{Solver}}(s,a) = \tau\, \log \pi_{Solver}(a|s). \tag{15}$$

Proof. Taking the logarithm of Equation 14 gives $\log \pi_{\text{Solver}}(a|s) = \frac{1}{\tau}\big(Q^{\pi_{\text{Solver}}}(s,a) - V^{\pi_{\text{Solver}}}(s)\big) = \frac{1}{\tau} A^{\pi_{\text{Solver}}}(s,a)$. Multiplying both sides by τ yields Equation 15. □

Intuition. Under a soft-optimal policy, how much better an action is than average ($A^{\pi_{\text{Solver}}}$) and how strongly the teacher prefers it ($\log \pi_{\text{Solver}}$) are the same quantity up to the temperature τ. This translates the advantage signal into a teacher log-preference signal.

Lemma C.6 (Shaping Reduces to Scaled Log-Probability)**.** *Under Assumptions C.1 and C.2, the shaped solver term in Equation 8 satisfies:*

$$\alpha\, h\big(\widetilde{A}_{i,t}^{\pi_{Solver}}\big) = \beta\, \log \pi_{Solver}(a_{i,t}|s_{i,t}) + c_0 + O\big((A^{\pi_{Solver}})^2\big), \tag{16}$$

where $\beta := \alpha\tau/\big(\sqrt{2}\,(\mathrm{RMS}_{\mathcal{B}}(g)+\epsilon)\big) > 0$ is a batch-level constant and c_0 is an action-independent constant that vanishes from the policy gradient by baseline invariance.

Proof. Recall that $\widetilde{A} = A^{\pi_{\text{Solver}}} + 1$ (Equation 5) and $h(x) = g(x)/(\text{RMS} + \epsilon)$ with $g = \operatorname{asinh}$ (Equation 7). Write $\beta' = \alpha/(\text{RMS} + \epsilon)$, so $\alpha\, h(\widetilde{A}) = \beta' \operatorname{asinh}(A + 1)$. Under Assumption C.2, Taylor-expand around the optimal-move point $A = 0$ (i.e., $\widetilde{A} = 1$). Using $\operatorname{asinh}'(x) = 1/\sqrt{x^2+1}$:

$$\operatorname{asinh}(A+1) = \operatorname{asinh}(1) + \frac{1}{\sqrt{1^2+1}} \cdot A + O(A^2) = \operatorname{asinh}(1) + \frac{A}{\sqrt{2}} + O(A^2).$$

Thus $\alpha\, h(\widetilde{A}) = \underbrace{\beta' \operatorname{asinh}(1)}_{c_0} + \frac{\beta'}{\sqrt{2}} A + O(A^2)$. By Lemma C.5, $A = \tau \log \pi_{\text{Solver}}$, so the action-dependent term becomes $\frac{\beta'\tau}{\sqrt{2}} \log \pi_{\text{Solver}} = \beta \log \pi_{\text{Solver}}$. □

Intuition. The asinh function, expanded around the optimal-move baseline $\widetilde{A} = 1$, is locally linear in $A^{\pi_{\text{Solver}}}$. Combined with the batch-constant RMS normalization, the shaped signal reduces to $\beta \cdot \log \pi_{\text{Solver}}$, a constant coefficient times the teacher's log-preference.

C.5.4 Main Theorem

We now prove Theorem 2.1 from Section 2, restated here for convenience.

Theorem 2.1 (Implicit Objective of CAST)**.** Under Assumptions C.1–C.4, the policy-gradient update direction of the training rule Equation 8 equals the gradient of:

$$\mathcal{J}_{\text{ours}}(\theta) = \mathbb{E}_{s_0\sim\mu}\big[V_{\text{task}}^{\pi_\theta}(s_0)\big] - \beta\, \mathbb{E}_{s\sim d^{\pi_\theta}}\big[\mathrm{H}\big(\pi_\theta(\cdot|s),\, \pi_{\text{Solver}}(\cdot|s)\big)\big],\ \beta = \frac{\alpha\,\tau}{\sqrt{2}\,(\mathrm{RMS}_{\mathcal{B}}(g)+\epsilon)}. \tag{17}$$

Proof. We assemble the lemmas in four steps.

Step 1 (Additivity). The training rule uses $\hat{A}_{i,t} = \hat{A}_i^{\text{outcome}} + \alpha\, h(\widetilde{A}_{i,t}^{\pi_{\text{Solver}}})$ as the advantage weight. Since the policy gradient is linear in the advantage weight, the update decomposes as $g_{\text{ours}} = g_{\text{GRPO}} + g_{\text{shape}}$.

Step 2 (GRPO → task return). By Assumption C.3 and the policy gradient theorem (Sutton et al., 1999):

$$g_{\text{GRPO}} \propto \nabla_\theta\, \mathbb{E}_{s_0\sim\mu}[V_{\text{task}}^{\pi_\theta}(s_0)].$$

Step 3 (Shaping → negative cross-entropy). By Lemma C.6, the shaping weight reduces to $\beta \log \pi_{\text{Solver}}(a|s)$ plus an action-independent constant. Since $\log \pi_{\text{Solver}}$ does not depend on θ, by Assumption C.4 the shaping term is the gradient of a surrogate:

$$g_{\text{shape}} = \nabla_\theta\Big(\beta\, \mathbb{E}_{s\sim d^{\pi_\theta},\, a\sim\pi_\theta}\big[\log \pi_{\text{Solver}}(a|s)\big]\Big) = -\nabla_\theta\Big(\beta\, \mathbb{E}_{s\sim d^{\pi_\theta}}\big[\mathrm{H}\big(\pi_\theta(\cdot|s),\, \pi_{\text{Solver}}(\cdot|s)\big)\big]\Big),$$

where the last equality uses the definition of cross-entropy Equation 12: $\mathbb{E}_{a\sim\pi_\theta}[\log \pi_{\text{Solver}}(a|s)] = -\mathrm{H}(\pi_\theta, \pi_{\text{Solver}})$.

Step 4 (Combine). Summing Steps 2 and 3 directly yields Equation 17. □

Reading the theorem. GRPO provides the "do the task right" gradient; the shaped solver advantage provides the "act like the teacher" gradient. The teacher's signal enters through the cross-entropy $\mathrm{H}(\pi_\theta, \pi_{\text{Solver}})$, the standard objective for knowledge distillation (Agarwal et al., 2024). By the

decomposition Equation 13, minimizing the cross-entropy is equivalent to simultaneously minimizing $\mathrm{KL}(\pi_\theta \| \pi_{\text{Solver}})$ (move toward the solver) and minimizing $\mathcal{H}(\pi_\theta)$ (reduce entropy). The latter makes the objective *mode-seeking*: the student concentrates probability on the solver's preferred actions rather than spreading mass across all modes. This is the natural behavior of reverse-KL distillation.

Remark C.7 (Relation to KL-regularized objectives). Defining $J(\theta) = \mathbb{E}_\mu[V^{\pi_\theta}_{\text{task}}] - \beta\, \mathbb{E}_{d^{\pi_\theta}}[\mathrm{KL}(\pi_\theta \| \pi_{\text{Solver}})]$, the implicit objective can equivalently be written as

$$\mathcal{J}_{\text{ours}}(\theta) = J(\theta) - \beta\, \mathbb{E}_{d^{\pi_\theta}}[\mathcal{H}(\pi_\theta)]. \tag{18}$$

The additional $-\beta\mathcal{H}$ relative to the standard KL-regularized objective $J(\theta)$ is not a defect; it arises because the advantage weight $\log \pi_{\text{Solver}}$ in our training rule does not include a $-\log \pi_\theta$ term. Had the weight been $\log(\pi_{\text{Solver}}/\pi_\theta) = \log \pi_{\text{Solver}} - \log \pi_\theta$ instead, the surrogate would equal $-\mathrm{KL}(\pi_\theta \| \pi_{\text{Solver}})$ and $\mathcal{J}_{\text{ours}}$ would reduce to $J(\theta)$. Equivalently, adding a standard entropy bonus $+\beta\mathcal{H}$ to the training objective would recover $J(\theta)$ exactly, but our experiments show that the mode-seeking behavior of the cross-entropy form already works well in practice.

C.5.5 COROLLARIES

Corollary C.8 (Closed-Form Optimal Policy)**.** *Fix the state visitation distribution. The policy that maximizes the per-state KL-regularized objective $\mathcal{L}(\pi) = \mathbb{E}_{a\sim\pi}[A^{\pi_\theta}_{task}(s,a)] - \beta\, \mathrm{KL}(\pi \| \pi_{Solver})$ subject to $\sum_a \pi(a) = 1$ is:*

$$\pi^*(a|s) \propto \pi_{Solver}(a|s)\, \exp\Big(\frac{1}{\beta}\, A^{\pi_\theta}_{task}(s,a)\Big). \tag{19}$$

Proof. Introduce a Lagrange multiplier λ for the normalization constraint:

$$\mathcal{L} = \textstyle\sum_a \pi(a) A_{\text{task}} - \beta \sum_a \pi(a) \log \frac{\pi(a)}{\pi_{\text{Solver}}(a)} + \lambda\big(1 - \sum_a \pi(a)\big).$$

Setting $\partial\mathcal{L}/\partial\pi(a) = 0$ gives $\log \frac{\pi(a)}{\pi_{\text{Solver}}(a)} = \frac{1}{\beta} A_{\text{task}}(s,a) - 1 - \frac{\lambda}{\beta}$. Exponentiating and absorbing the a-independent terms into a normalization constant yields Equation 19. The second-order condition $\partial^2\mathcal{L}/\partial\pi(a)^2 = -\beta/\pi(a) < 0$ confirms this is a maximum. □

Why the student can surpass the solver. Equation Equation 19 has the form *posterior* $\propto$ *prior* $\times$ *likelihood*: the solver distribution serves as a prior, and the task advantage tilts it exponentially. In the pure-distillation limit $\beta \to \infty$, the tilt vanishes and $\pi^* \to \pi_{\text{Solver}}$. At finite β, the factor $\exp(A_{\text{task}}/\beta)$ lets π^* deviate wherever the task reward warrants it, enabling the student to *surpass* the teacher.

Remark C.9. The corollary solves the KL-regularized subproblem, which is the natural trust-region objective (Schulman et al., 2017). Since the cross-entropy $\mathrm{H}(\pi, \pi_S) = -\sum_a \pi \log \pi_S$ is *linear* in π, the cross-entropy-regularized problem $\max_\pi \mathbb{E}_\pi[A_{\text{task}}] - \beta\, \mathrm{H}(\pi, \pi_S)$ has no interior optimum over the simplex. Instead, Theorem 2.1 shows the implicit objective decomposes into a KL penalty and an entropy reduction; the KL-regularized subproblem isolates the component with a well-defined closed-form solution.

Corollary C.10 (Full-Domain Robustified Distillation)**.** *Dropping Assumption C.2 and retaining the* asinh *nonlinearity, define the step-varying effective gain:*

$$\beta_{eff}(A^{\pi_{Solver}}) = \frac{\alpha\, \tau}{\mathrm{RMS}_{\mathcal{B}}(g) + \epsilon} \cdot \frac{\mathrm{asinh}(A^{\pi_{Solver}} + 1) - \mathrm{asinh}(1)}{A^{\pi_{Solver}}}. \tag{20}$$

Then Theorem 2.1 holds with β replaced by β_{eff}: the constant distillation coefficient becomes step-dependent, automatically down-weighting outlier steps. As $A^{\pi_{Solver}} \to 0$, $\beta_{eff} \to \beta$ (by L'Hôpital's rule, the ratio tends to $\mathrm{asinh}'(1) = 1/\sqrt{2}$), recovering Theorem 2.1.

Proof. Without the Taylor truncation, the action-dependent part of $\alpha\, h(\widetilde{A})$ is $\beta'\big(\mathrm{asinh}(A+1) - \mathrm{asinh}(1)\big)$ (the constant $\beta'\,\mathrm{asinh}(1)$ vanishes by baseline invariance). Factoring out A:

$$\beta'\, \frac{\mathrm{asinh}(A+1) - \mathrm{asinh}(1)}{A} \cdot A = \frac{\beta_{\text{eff}}}{\tau} \cdot \tau \log \pi_{\text{Solver}}$$

where the last step uses Lemma C.5. Substituting β_{eff} for β in Step 3 of the theorem proof, the remaining steps are identical. For large $|A|$, β_{eff} decreases because asinh grows only logarithmically while A grows linearly, so the ratio shrinks. □

Intuition. The asinh compression causes the effective distillation strength to decrease for steps with unusually large solver advantages (e.g., dead-state penalties), matching the design intent of soft-compressing extreme values.

C.5.6 Performance Improvement via the Performance Difference Lemma

We now use the Performance Difference Lemma (Kakade & Langford, 2002) to give a complementary perspective: the solver advantage used as a per-step reward defines an *augmented return* whose maximization provably improves over the solver, with a sub-optimality bound that scales linearly in the horizon H rather than quadratically.

Proposition C.11 (Performance Difference Lemma (Kakade & Langford, 2002))**.** *For any two policies π, π' and any reward function r in a finite-horizon MDP:*

$$V_r^{\pi}(s_0) - V_r^{\pi'}(s_0) = \sum_{t=0}^{H-1} \mathbb{E}_{s_t \sim d_t^{\pi}}\Big[\mathbb{E}_{a\sim\pi(\cdot|s_t)}\big[A_r^{\pi'}(s_t, a)\big]\Big]. \tag{21}$$

The PDL states that the value gap between two policies equals the cumulative advantage of the new policy π evaluated under the old policy π''s value function, but weighted by the new policy's own state distribution. It is a standard result in RL theory; we state it without proof.

Proposition C.12 (Endogenous Reward and Reward Shaping)**.** *Under Assumption C.1, define the solver's* endogenous reward *(Li et al., 2025):*

$$\hat{r}(s_t, a_t) = \tau \log \pi_{\textit{Solver}}(a_t|s_t) + V^{\pi_{\textit{Solver}}}(s_t) - V^{\pi_{\textit{Solver}}}(s_{t+1}). \tag{22}$$

This is a potential-based reward shaping of the base signal $\tilde{r}(s, a) = \tau \log \pi_{\textit{Solver}}(a|s)$ with potential $\Phi(s) = V^{\pi_{\textit{Solver}}}(s)$. By the reward shaping theorem (Ng et al., 1999), potential-based shaping preserves the optimal policy: $\hat{r}$ and $\tilde{r}$ induce the same policy ordering and the same policy gradient.

Proof. Write $\hat{r}(s_t, a_t) = \tilde{r}(s_t, a_t) + \Phi(s_t) - \Phi(s_{t+1})$. The added term $\Phi(s_t) - \Phi(s_{t+1})$ is a potential-based shaping function: it depends only on s_t and s_{t+1} and uses the potential $\Phi(s) = V^{\pi_{\text{Solver}}}(s)$. By the reward shaping theorem (Ng et al., 1999, Theorem 1), potential-based shaping preserves the optimal policy and the policy ordering: the optimal policy under $\hat{r}$ and $\tilde{r}$ is the same, and both rewards induce the same policy gradient (up to a state-dependent baseline). □

Connection to the implicit objective. By Lemma C.6, our training rule uses $\beta\tilde{r}(s, a)/\tau = \beta \log \pi_{\text{Solver}}(a|s)$ as the per-step advantage weight. Proposition C.12 shows that this base signal $\tilde{r} = \tau \log \pi_{\text{Solver}}$ is related to the endogenous reward $\hat{r}$ by potential-based shaping, so they share the same policy gradient. In our deterministic games with the shortest-path solver reward $r_S = -1$ per step, a direct calculation shows that the endogenous reward reduces to $\hat{r}(s_t, a_t) = r_S = -1$ for all (s_t, a_t): substituting $A^{\pi_{\text{Solver}}} = (r_S + V^{\pi_{\text{Solver}}}(s_{t+1})) - V^{\pi_{\text{Solver}}}(s_t)$ into Equation 22, the value-function terms telescope and only the base reward $r_S = -1$ remains. This confirms that the action-quality information resides entirely in the base signal $\tilde{r} = \tau \log \pi_{\text{Solver}} = A^{\pi_{\text{Solver}}}$, which by Lemma C.5 equals the solver advantage, and the shaping potential plays no role in the policy gradient.

Performance improvement guarantee. Applying the PDL (Proposition C.11) with $\pi = \pi_\theta$ and $\pi' = \pi_{\text{Solver}}$ under the augmented reward $r_{\text{aug}} = r_{\text{task}} + (\beta/\tau) \cdot \tilde{r}$:

$$V_{r_{\text{aug}}}^{\pi_\theta}(s_0) - V_{r_{\text{aug}}}^{\pi_{\text{Solver}}}(s_0) = \sum_t \mathbb{E}_{d_t^{\pi_\theta}, \pi_\theta}\big[A_{r_{\text{aug}}}^{\pi_{\text{Solver}}}(s_t, a_t)\big]. \tag{23}$$

By Theorem 2.1, our training rule performs gradient ascent on $\mathcal{J}_{\text{ours}}$, which is precisely the augmented return $V_{r_{\text{aug}}}^{\pi_\theta}$ (up to constant boundary terms from reward shaping). The PDL guarantees that whenever

$\mathbb{E}_{\pi_\theta}[A^{\pi_{\text{Solver}}}_{r_{\text{aug}}}(s,\cdot)] \geq 0$ at every state, the student's augmented value improves over the solver's, i.e., $V^{\pi_\theta}_{r_{\text{aug}}} \geq V^{\pi_{\text{Solver}}}_{r_{\text{aug}}}$. Moreover, following the analysis of Li et al. (2025), if the solver is an ϵ_π-approximate expert (in the sense that $\max_{s,a} |\log \pi_{\text{Solver}}(a|s) - \log \pi^*(a|s)| \leq \epsilon_\pi$ for the true optimal policy π^*), the sub-optimality of π_θ under the task reward scales as $O(H\epsilon_\pi)$, improving over the $O(H^2\epsilon_\pi)$ bound of behavioral cloning. The linear dependence on H arises because the reward-shaping structure of $\hat{r}$ eliminates the compounding-error problem inherent in imitation learning (Ross et al., 2011).

C.5.7 Discussion of Assumptions

(A1) Soft-optimal solver. Lemma C.5 requires the solver to be a soft-optimal policy (Haarnoja et al., 2018). When the solver deviates from optimality, the distillation interpretation degrades proportionally: π_{Solver} should be understood as the solver's empirical behavior distribution. For our three games, the solvers are near-optimal, making this assumption well-justified.

(A2) Small signal. The linearization of $\operatorname{asinh}(A+1)$ around $A=0$ is accurate when $|A^{\pi_{\text{Solver}}}| \lesssim 1$, which holds for the vast majority of moves. For the rare large-magnitude steps (e.g., transitions to dead states), Corollary C.10 provides the exact characterization via β_{eff}: the qualitative conclusion is unchanged, only the distillation coefficient becomes step-dependent.

(A3) GRPO consistency. The group-relative baseline (Shao et al., 2024) introduces finite-sample variance but does not bias the gradient direction. This affects the precision of the task-return term but not the structural form of the implicit objective.

(A4) Frozen visitation surrogate. Treating the shaping term as a per-step immediate weight and freezing the visitation distribution omits the long-range term $\nabla_\theta d^{\pi_\theta}$, the same approximation used in PPO surrogate objectives (Schulman et al., 2017). This is consistent with how the method applies the solver signal on a per-turn basis in practice.

C.6 Training and Evaluation Details

C.6.1 Training Hyperparameters

All backbone RL algorithms we compare (GRPO, DAPO, and GSPO) share the same backbone model, training data, rollout budget, and optimizer configuration, and differ only in their policy-optimization objective. CAST uses the DAPO objective, while GiGPO is implemented as a separate process-level baseline. We list the shared configuration in Table 5, the objective-specific hyperparameters in Table 6, and the GiGPO-specific hyperparameters in Table 7.

Table 5: Shared training configuration used by all algorithms.

Hyperparameter	Value
Backbone model	Qwen3-4B-Instruct-2507
Learning rate	1e-6
Train batch size	16
Rollouts per prompt (G)	8
Training updates (Sokoban / Minesweeper / Rush Hour)	200 / 400 / 300

C.6.2 Evaluation Settings

We evaluate every model under a single protocol across the three games. Each instance is run as the same multi-turn agent loop used in training (Appendix C.1), and an episode is scored with the sparse $0/1$ environment reward: an instance counts as solved only if the agent reaches the goal within the per-game turn budget. For each game, we report both the in-domain (ID) tier and the harder unseen-difficulty (Unseen) tier of Table 3, evaluating on the 200-instance test split of each tier. Every instance is attempted with 4 independent rollouts.

Table 6: Objective-specific hyperparameters for each backbone RL algorithm. The GSPO clip thresholds are much smaller than those of GRPO/DAPO because GSPO clips at the sequence-level importance ratio rather than the token level, so the two are not on the same scale.

Hyperparameter	**GRPO**	**DAPO**	**GSPO**
Loss mode	`vanilla`	`vanilla`	`gspo`
Loss aggregation	`token-mean`	`token-mean`	`seq-mean-token-mean`
KL loss coefficient	0.001	0.0	0.0
Clip ratio (low)	0.2	0.2	0.0003
Clip ratio (high)	0.2	0.28	0.0004
Dual-clip constant c	3.0	3.0	3.0
Dynamic sampling (filter groups)	Off	Off	Off

Table 7: GiGPO-specific hyperparameters for the process-level baseline.

Hyperparameter	**Value**
GiGPO enabled	Yes
Step-advantage weight	1.0
Step-advantage normalization	`mean_std_norm`
Discount factor γ	0.95

Open-source models. Open-source backbones are served locally with SGLang and decoded with temperature 0.6 and top-p 0.95. We use a cumulative response budget of $16{,}384$ tokens per episode and a configured initial prompt limit of $2{,}048$ tokens. The per-episode turn budget is game- and split-specific: 30 turns for Sokoban and Minesweeper ID and 40 for their Unseen tier, and 20 turns for Rush Hour on both tiers.

Table 8: Evaluation settings for open-source models, taken from the evaluation configs. The per-episode turn budget varies by game and split.

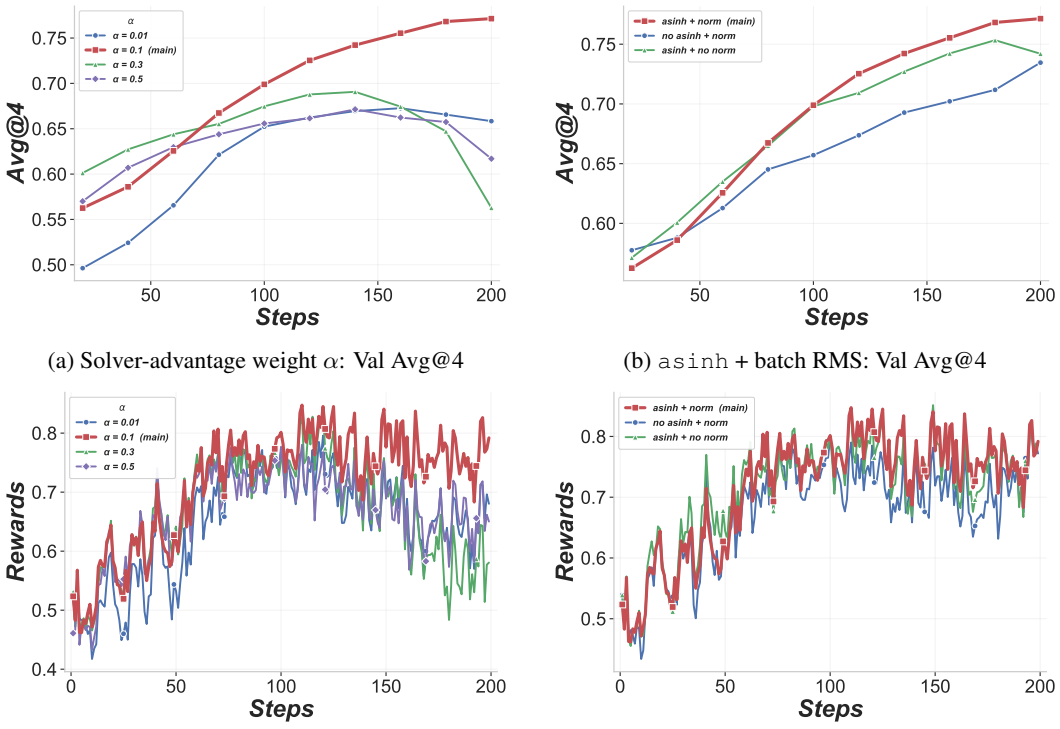
Setting	**Value**
Inference engine	SGLang
Temperature	0.6
Top-p	0.95
Response-token budget per episode	16,384
Configured initial prompt limit	2,048
Rollouts per instance	4
Turn budget (Sokoban / Minesweeper)	30 (ID) / 40 (Unseen)
Turn budget (Rush Hour)	20

Closed-source models. Closed-source models are queried through an OpenAI-compatible API under the same interaction protocol, per-game turn budgets, and $0/1$ scoring as the open-source setting. We evaluate six proprietary models: Claude Sonnet 4.5, Claude Sonnet 4.6, Claude Opus 4.5, Claude Opus 4.6, Gemini 2.5 Pro, and Gemini 2.5 Flash, with temperature 1.0, top-p 1.0, and 4 rollouts per instance. We use the same configured episode-level response budget and initial-prompt setting as above.

D Ablation Study Curves

Table 9: Evaluation settings for closed-source models. The interaction protocol, per-episode turn budgets, and $0/1$ scoring match the open-source setting (Table 8).

Setting	Value
Serving interface	OpenAI-compatible API
Temperature	1.0
Top-p	1.0
Response-token budget per episode	16,384
Configured initial prompt limit	2,048
Rollouts per instance	4
Turn budget (Sokoban / Minesweeper)	30 (ID) / 40 (Unseen)
Turn budget (Rush Hour)	20



(a) Solver-advantage weight α: Val Avg@4

(b) `asinh` + batch RMS: Val Avg@4

(c) Solver-advantage weight α: Train rewards

(d) `asinh` + batch RMS: Train rewards

Figure 6: **Ablation studies on Sokoban. Left column**: sweeping the solver-advantage weight α. **Right column**: removing/replacing the `asinh` transformation and batch-level RMS normalization. Curves are EMA-smoothed at 0.6.