WITNESS: DISCOVERY, DECIPHERING, AND EPIPHANY IN INTERACTIVE PUZZLE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated science needs agents that can work out the rules of an unfamiliar environment by interacting with it. Interactive rule-discovery puzzles offer a controlled setting for studying this ability: an agent infers hidden rules through experimentation and uses what it has inferred to reach a stated goal. We ask what limits current language models on these puzzles and whether reinforcement learning (RL) improves performance on rules held out from training. To study both, we introduce WITNESS, a 2D grid-based puzzle environment with ground-truth ASCII observations and controlled access to rules. An agentic pipeline generates games for WitnessGym, the RL training suite, and WitnessBench, comprising public validation and private test games. The validation set separately tests new compositions of trained rule primitives and primitives absent from training. Under a shared harness, the best of 18 frontier proprietary and open-weight models solves only 24% of private test level slots, with scores sensitive to the observation interface and agent configuration. Providing ground-truth rules raises Opus-5's validation RHAE-L5 (relative human action efficiency over the first five levels) from 59.9 to 97.8, whereas a 27B open-weight model gains only 2.1 points and remains limited even with the rules provided. RL on WitnessGym raises the 27B model's private test RHAE-L5 from 2.1 to 5.4 and yields a mean gain of 4.1 points on four external discovery benchmarks. Together, these results point to rule acquisition as a major difficulty for frontier models like Opus-5 while smaller models further struggle on rule-based execution, and indicate that RL on hidden-rule puzzles transfers to broader rules and real-world tasks beyond training. Benchmark, training split, and scripts are provided in supplementary materials.

1 INTRODUCTION

Discovering the rules governing an environment requires an agent to form hypotheses, choose informative actions, and revise its beliefs based on the resulting observations (King et al., 2009; Cook et al., 2011). Interactive rule-discovery puzzles provide a controlled setting for studying this process: agents interpret observations, gather evidence about hidden rules, and use that evidence to guide their actions toward a goal. However, these capabilities are entangled: A low success rate does not by itself indicate whether state interpretation, rule discovery, or planning failed. We therefore ask two questions: where current models struggle, and, beyond diagnosing limitations, whether reinforcement learning (RL) improves performance on rules held out from training (Fawzi et al., 2022).

Addressing these questions requires two kinds of control: for diagnosis, control over the observation interface and over access to ground-truth rules; for transfer, training and evaluation sets separated by rule. ARC-AGI-3 (ARC Prize Foundation, 2026a), the closest benchmark in spirit, evaluates rule discovery in visually presented games, where performance depends on both state interpretation and discovery. Text-based benchmarks avoid the visual gap but drop something else: DiG-Bench (Battleday et al., 2026) poses rule discovery over a single-line string with no spatial structure, and MazeBench (Pappas & Pappas, 2026) has spatial structure but rules are fixed and known to the model. Other game benchmarks (Chen et al., 2026c; Li et al., 2026; Hu et al., 2026; Verma et al., 2025; Chen et al., 2026a) test reasoning over static patterns or over games whose rules are given, and failures on new puzzle variants (Liang et al., 2026) raise concerns about generalization beyond those families.

To provide both kinds of control, we introduce WITNESS (Figure 1), an execution environment for 2D grid-based puzzles in the spirit of *The Witness* (Thekla, Inc., 2016). Each game asks the agent to

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

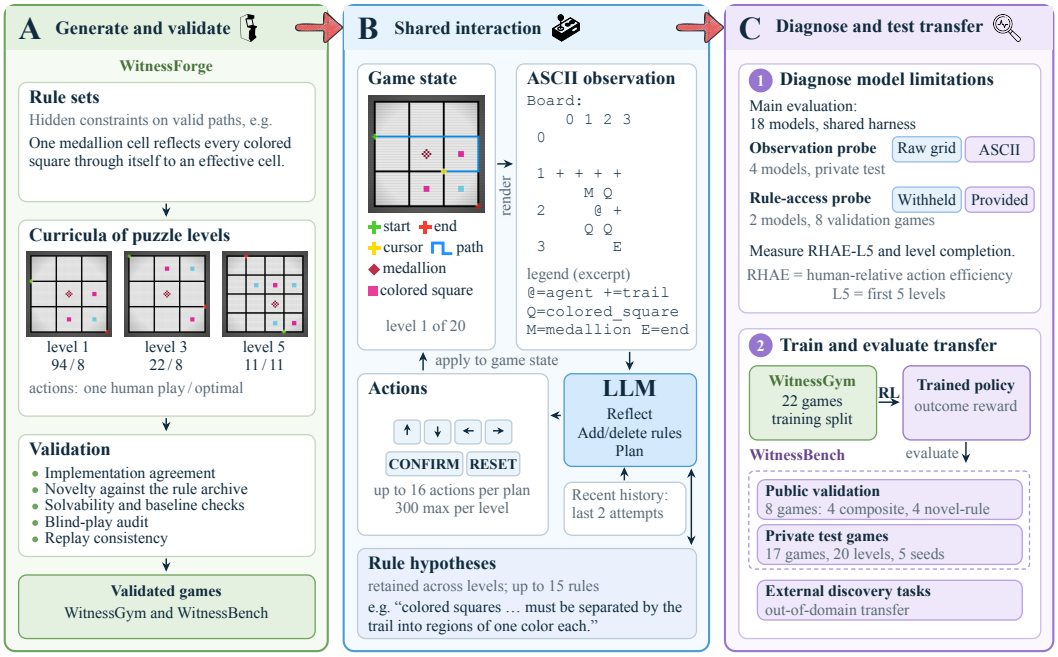


Figure 1: Overview of **W**itness-**I**nspired **T**estbed of **N**ovel **E**nvironments with **S**ymbolic **S**tructure (WITNESS).

draw a path from a start to a goal on a grid under hidden rules that constrain which paths are valid, and the engine exposes the board as a ground-truth ASCII representation, so the model parses spatial structure without visual recognition. An agentic pipeline synthesizes new games with their own curricula of levels, from which we build WitnessGym for training and WitnessBench for evaluation, with public validation and private test sets. To distinguish compositional transfer from transfer to new rule primitives, we organize the validation games by their relationship to the RL training rules: held-out compositions (new combinations of trained primitives) and held-out primitives (rules whose primitives do not appear in training).

On the first question, diagnosis in this environment under a shared harness indicates that acquiring the rules is a major difficulty for Opus-5, while substantial difficulty remains for Qwen3.8-27B even when the rules are provided. Under a harness that standardizes observations, memory, and actions and permits no code execution, the best of 18 frontier proprietary and open-weight models, Fable-5, solves only 24% of private test level slots. To assess score dependency on observation interface, we replace ASCII rendering with the raw grid, which substantially reduces performance for several models while leaving Opus-5 largely unaffected. In a separate rule-access probe, providing ground-truth rules raises Opus-5's validation relative human action efficiency over the first five levels (ARC Prize Foundation, 2026b) (RHAE-L5; higher is better) from 59.9 to 97.8, with all five evaluated levels solved in each of the 40 game runs. For Qwen3.8-27B, the improvement is smaller, from 8.0 to 10.1 RHAE-L5.

The second question is whether RL on games with hidden rules improves performance on rules held out from training. To answer this question, we train Qwen3.8-27B in WitnessGym using RL with outcome-based reward and evaluate its transfer to held-out games and external tasks. The trained model improves from 2.1 to 5.4 RHAE-L5 on the private test set with thinking off, and achieves higher validation performance on both held-out compositions and held-out primitives. Beyond the suite, the trained model achieves a mean gain of 4.1 percentage points (95% CI 2.1 to 6.2) on four external benchmarks of discovery-related reasoning (Li et al., 2025; Chen et al., 2026b; Banatt et al., 2024; Jhaveri et al., 2026), suggesting that part of what is learned is not specific to these puzzles. We observe no improvement for Qwen3.5-9B under the same training setup, and credit assignment keyed to surviving rule hypotheses did not yield additional gains over broadcasting episode advantages.

Taken together, these diagnostic and training results support three contributions:

- WITNESS, an environment and generation pipeline for grid-based rule-discovery puzzles with a text interface, from which we release WitnessGym for training and WitnessBench for evaluation, with validation rules held out from RL training as held-out compositions and held-out primitives.

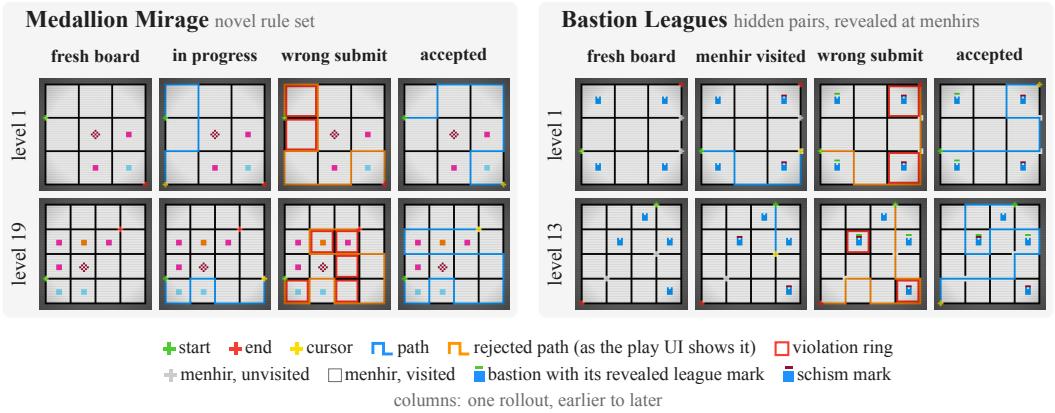


Figure 2: One level of a WITNESS game as the engine renders it and as the model observes it: the ASCII board with its legend, the current path, and the feedback shown after a rejected submission.

- An evaluation of 18 models under a shared harness, with controlled observation and rule-access probes indicating that acquiring the rules is a major difficulty for Opus-5 and that difficulty persists for Qwen3.8-27B when the rules are provided.
- Evidence that RL with outcome-based reward improves performance on the private test set, on held-out compositions and held-out primitives, and on four external benchmarks, together with an empirical comparison in which credit assignment keyed to surviving hypotheses did not yield additional gains over broadcasting episode advantages.

2 Benchmark

WITNESS comprises a game engine, a suite of games with WitnessBench for evaluation and WitnessGym for training, a pipeline that generates and validates new games, a shared evaluation harness, and an action-efficiency metric.

2.1 Game Engine and Mechanics

The engine is built on the official ARC-AGI-3 SDK (Foundation, 2026) and renders each frame as a 64×64 pixel image. It also emits a ground-truth ASCII representation of the board, which serves as the model's observation (Figure 2). The board is an $M \times N$ grid on which the agent draws a path from a start cell to a goal cell; a level is solved when the submitted path satisfies every rule of the game. The action space consists of four directional moves, SUBMIT, and RESET. The semantics of SUBMIT and RESET are stated to the model, but the mapping from the remaining action IDs to directions has to be inferred through interaction.

Game rules constrain which paths are valid, and discovering them is left to the agent. Classic games use publicly documented primitives from *The Witness*, while generated games use rules constructed and validated by WitnessForge (Section 2.3). The classic primitives may have appeared in model pretraining; throughout the paper, held-out status refers only to the RL training split.

The engine provides two kinds of feedback. An invalid move, such as crossing a wall or a broken edge, is refused and leaves the frame unchanged. A submitted path that violates a rule is rejected, with the path and violating cells highlighted in both the image and the ASCII representation. These highlights depend on the submitted path, allowing different attempts to reveal different evidence about the rule. Some games, such as *Bastion Leagues* in Figure 2, additionally require interactions that reveal hints: the agent has to discover how to elicit these hints before using them to plan a solution.

2.2 Game Suite: WitnessBench and WitnessGym

WitnessBench is the benchmark: 8 public validation games and 17 private test games, on which every model comparison in this paper is reported. WitnessGym is its training counterpart: 22 public games that share the engine and harness with the benchmark but none of its rules. Every private test game consists of 20 levels: the first five use small grids and introduce the rules; remaining levels use larger grids on which the difficulty shifts to planning and executing valid paths under rules already learned.

The training set contains 8 classic games, each based on one primitive rule of *The Witness*, and 14 generated games. The validation set tests generalization at two levels: 4 classic games combine training primitives in new ways (held-out *compositions*), and 4 generated games use primitives absent from training (held-out *primitives*). The composition split deliberately reuses training primitives, while the primitive split tests transfer beyond them. The 17 test games are generated with rules not used in training and are kept private; the training and validation games are public, and the generation pipeline (Section 2.3) supports periodic refresh of private set like LiveCodeBench (Jain et al., 2025).

2.3 Game Generation and Validation

WitnessForge constructs and validates the generated games in both suites. Given a rule concept in natural language, it produces a packaged game containing a rule implementation, a curriculum of levels, and metadata, extensible without manually implementing rules or constructing levels. Its foundation is a path-enumeration kernel written in Rust: given a board, it enumerates every simple path from a start node to the goal node, and a predicate layer applies the rule-checking functions to return the exact set of valid solutions. The pipeline has two stages: rule verification followed by level design and audit; Algorithm 1 in Appendix B.8 gives the full procedure and the filtering thresholds.

Rule verification. A candidate rule is admitted after passing a series of programmatic checks; besides cheap static checks, three semantic checks carry the argument. First, two implementations are generated independently from the same specification and compared over enumerated path spaces on test boards; counterexamples guide revision for a bounded number of rounds, and candidates whose implementations still disagree are rejected. Second, a solution-space ablation removes the rule and checks that the valid-solution set strictly expands, showing that the rule excludes otherwise valid paths on the tested boards. Third, a novelty check screens for distinctness by comparing each candidate against the existing rule archive. Rules that reduce strictly to path-length checks are pruned as degenerate. These checks provide evidence of distinctness relative to the tested boards and archived rules, rather than guaranteeing novelty relative to pretraining data.

Level design and audit. Levels are designed to distinguish the intended rule from a predefined set of incorrect candidates through differences in their valid-solution sets, with each incorrect candidate refuted by at least one level's stored solution. For each admitted rule, the pipeline generates candidate levels and retains those that pass structural validation, whose shortest solution meets a minimum action count, and on which removing the rule strictly enlarges the valid-solution set. It then filters out levels solved by the tested rule-agnostic heuristics or greedy baselines, or whose exact random-play success probability exceeds a fixed threshold. The retained levels are annotated with the candidates they refute and ordered into a curriculum that introduces the rule and eliminates remaining alternatives, generating additional levels when the pool lacks a discriminating one. The assembled game then undergoes three final checks. A held-out language model plays it blind to test whether its rule can be recovered from interaction alone; an independent audit re-implements all filters from scratch to probe levels with fresh strategies and seeds; and each stored solution is replayed against the packaged game to verify solvability. Across the released games, 5 of 42 candidate rules were rejected, most often for their unlearnability across models; 97% candidate levels were rejected by the programmatic filters, as random play could solve them or they admitted too many solutions, preserving 760 levels.

2.4 Evaluation Harness

All models interact with the engine through the same harness, which standardizes observations, memory support, and available actions. Models reason in text and interact with the game through six actions, without code execution or external tools. At each turn, the harness presents the current board as ASCII text with a symbol legend. Alongside the board, it provides the path drawn so far, the previous action and its effect, and a record of recent attempts on the current level and their outcomes. To support reasoning across attempts and levels, the observation also includes a rule memory containing hypotheses recorded by the model. This memory persists within each game and is cleared between games. Given this observation, the model responds with free-form analysis, updates to its rule memory (add, delete, or keep), and an optional sequence of up to 16 actions. The engine executes the sequence before returning the next observation, stopping early if the level is solved. If the model returns no sequence, the harness instead executes one random action, which counts toward the action budget like any other. Interaction continues until the final level is solved or a level remains unsolved after its budget of 300 actions per level or reaching 1000 LLM calls per game; an unsolved level ends the run rather than being skipped. Complete prompts are provided in Appendix B.6.

2.5 Evaluation Metric and Protocol

The evaluation metric is Relative Human Action Efficiency (RHAE), taken from ARC-AGI-3 (ARC Prize Foundation, 2026b). A solved level is scored by the agent's action count S^{ℓ}_{agent} against a reference count S^{ℓ}_{ref}, capped by 1.15; an unsolved or unreached level scores zero:

$$\text{score}_{\ell} = \begin{cases} \min\Big(\big(S^{\ell}_{\text{ref}}/S^{\ell}_{\text{agent}}\big)^2,\ 1.15\Big) & \text{if level } \ell \text{ is solved,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The reference count is the mean action count across successful plays by team members who had no access to the rule specification, counted on the same basis as the agent's actions, including failed attempts and resets. All levels in validation and test set games have successful human play data.

For a game g evaluated over its first L levels, the game score weights each level by its index ℓ and is capped at the weighted completion fraction:

$$\text{Score}_g \;=\; 100 \cdot \min\left(\frac{\sum_{\ell=1}^{L} \ell \cdot \text{score}_{\ell}}{\sum_{\ell=1}^{L} \ell},\ \frac{\sum_{\ell=1}^{L} \ell \cdot \mathbb{1}[\text{level } \ell \text{ solved}]}{\sum_{\ell=1}^{L} \ell} \right). \tag{2}$$

Because each level score is at most 1.15, the first term can exceed the second only when the weighted mean efficiency exceeds one; the minimum therefore allows above-reference efficiency on one level to offset below-reference efficiency on another, but never to offset an unsolved level. We report RHAE-L5 over the first five levels ($L = 5$) as the headline metric, emphasizing introductory performance, and RHAE-L20 over all twenty levels ($L = 20$) to measure full curriculum performance. The game engine stores the optimal step S^{ℓ}_{optim} for each level ℓ. Objective action efficiency (where $S^{\ell}_{\text{ref}} = S^{\ell}_{\text{optim}}$) and fully uncapped RHAE scores are also reported for reference and analysis.

For each seed, the dataset score is the mean game score across the split (17 test or 8 validation games). We report the mean and standard deviation across $M = 5$ test seeds and $M = 3$ validation seeds (except Table 5 in Appendix B.1, using 5 seeds for GT-rule ablation), giving 85 test runs per model.

3 Model Evaluation: What Limits Current Models

This section addresses the first question of the introduction, what limits current models, in three steps. Section 3.1 reports the overall performance of 18 models on the private test set under the shared harness. Section 3.2 compares this harness with alternative observation and harness settings to measure how much the scores depend on them. Section 3.3 measures the effect of providing the ground-truth rules on the validation set, to isolate the rule acquisition difficulty.

3.1 Overall Performance

Table 1 reports 18 proprietary and open-weight models on the 17 private test games, with 5 seeds per game, giving 85 runs and 1,700 level slots per model. All models use the harness of Section 2.4 and the scoring rule of Section 2.5; each model runs at its highest reasoning setting, with per-model configurations and full per-model statistics in appendices. Two findings stand out from the table.

First, under the shared harness, all evaluated models solve only a minority of the available level slots, and complete runs are rare. The model with the highest mean RHAE-L5, Fable-5, reaches 36.4 ± 3.7, solves 414 of 1,700 level slots (4.87 levels per run), and completes 2 of 85 runs. The lowest-scoring model, Qwen3.5-9B, reaches 1.0 ± 0.8, solves 44 level slots (0.52 levels per run), and completes no run. Twelve of the 18 models complete no run. Qwen3.8-27B, the starting checkpoint used for training in Section 4, reaches an RHAE-L5 of 6.4 at its highest reasoning setting; Section 4 evaluates this model with thinking off, so its numbers there are not comparable to this table.

Second, the ranking depends on the evaluation window. GPT-6-Astra-Pro has the highest RHAE-L20 at 12.8 ± 1.1 and completes 10 of 85 runs, whereas Fable-5 scores 8.3 on RHAE-L20 with 2 completed runs. Performance on the first five levels and performance over the full curriculum therefore give different orderings, and a higher early score does not imply more completed runs. We report both windows; Sections 3.2 and 3.3 examine what the scores depend on.

Table 1: **Model Evals using our canonical harness** (unseen games w/ novel rules, 20 lvls/game, 5 eval seeds). RHAE-L5 / RHAE-L20: RHAE computed over the first 5 levels (the benchmark number) and over all 20 levels. The first 5 levels are four rule-teaching levels with curriculum design plus a non-teaching testing level; the remaining levels are more complex levels of larger grid or harder. RHAE-uncap: the human-baseline score without the two caps. OAE (objective action efficiency): the same level-weighted efficiency score against the optimal action count. Deepest level: the most levels completed in any single rollout. Evaluated in **max-effort** where the endpoint exposed effort tiers.

Model	RHAE-L5 mean ± std	Beaten levels	Avg levels per game	OAE L5 / L20	RHAE-uncap L5 / L20	Deepest level	Beaten games	RHAE-L20 mean ± std
Fable-5	36.4 ± 3.7	414/1700(24%)	4.87/20	20.9 / 5.4	156.3 / 34.3	20/20	2/85	8.3 ± 2.7
Opus-5	30.6 ± 2.0	405/1700(24%)	4.76/20	21.2 / 6.6	210.9 / 38.6	20/20	2/85	9.4 ± 2.2
GPT-6-Astra-Pro	29.8 ± 2.0	399/1700(23%)	4.69/20	19.5 / 9.1	224.1 / 59.5	20/20	10/85	12.8 ± 1.1
GPT-6-Astra	27.6 ± 0.9	389/1700(23%)	4.58/20	17.6 / 7.7	182.7 / 50.0	20/20	10/85	11.3 ± 0.9
Kimi-K3	24.6 ± 4.0	249/1700(15%)	2.93/20	14.9 / 1.8	119.2 / 15.2	20/20	1/85	3.1 ± 1.6
GPT-5.6-Sol	23.0 ± 3.1	262/1700(15%)	3.08/20	13.8 / 2.1	92.7 / 10.8	14/20	0/85	3.4 ± 1.0
Opus-4.8	22.8 ± 4.3	203/1700(12%)	2.39/20	14.4 / 1.1	106.2 / 7.8	9/20	0/85	1.8 ± 0.3
Muse-Spark-1.2	14.9 ± 0.4	194/1700(11%)	2.28/20	8.2 / 0.7	28.2 / 2.3	11/20	0/85	1.2 ± 0.1
Grok-4.6	12.9 ± 2.6	240/1700(14%)	2.82/20	5.8 / 0.7	43.2 / 4.8	20/20	1/85	1.6 ± 1.1
DeepSeek V4 Pro	7.1 ± 2.3	132/1700(8%)	1.55/20	4.0 / 0.3	47.8 / 3.4	5/20	0/85	0.5 ± 0.2
Qwen-3.8-Max	6.9 ± 1.0	132/1700(8%)	1.55/20	4.0 / 0.3	26.8 / 1.9	5/20	0/85	0.5 ± 0.1
Qwen3.8-27B	6.4 ± 1.2	112/1700(7%)	1.32/20	3.9 / 0.3	16.8 / 1.2	5/20	0/85	0.5 ± 0.1
GLM-5.2-Max	5.6 ± 0.8	109/1700(6%)	1.28/20	2.8 / 0.2	20.8 / 1.5	5/20	0/85	0.4 ± 0.1
GLM-5.3	4.7 ± 2.6	90/1700(5%)	1.06/20	2.7 / 0.2	9.1 / 0.7	5/20	0/85	0.3 ± 0.2
Qwen3.5-35B-A3B	2.8 ± 1.3	58/1700(3%)	0.68/20	1.7 / 0.1	3.6 / 0.3	4/20	0/85	0.2 ± 0.1
Glim-30	2.7 ± 0.7	57/1700(3%)	0.67/20	1.9 / 0.1	5.6 / 0.4	4/20	0/85	0.2 ± 0.1
Dots-3-Note-Preview	2.6 ± 1.7	54/1700(3%)	0.64/20	0.9 / 0.1	10.6 / 0.8	5/20	0/85	0.2 ± 0.1
Qwen3.5-9B	1.0 ± 0.8	44/1700(3%)	0.52/20	0.6 / 0.0	1.4 / 0.1	3/20	0/85	0.1 ± 0.1

Table 2: **Harness ablation on the 17 unseen games** (5 seeds): how much of each model's RHAE-L5 score depends on its harness. *Our harness*: our full harness. *w/o perception*: same harness without ground-truth ASCII board (64×64 pixel input). *Minimal harness*: a minimalist loop, raw pixel input. *Prime-Agent*: A self-improving RLM Harness (95% on ARC-AGI-3 public games). *Model-attended solving* and *Verbal-only solving*: exclude cases – no model participation in level solving; winning moves computed by code. Details in Appendix B.5. *These models with Prime-Agent caused frequent refusals from Claude's content filter; we report 2 seeds to show a spread and repeat no further.

Harness	Opus-5	Opus-4.8	Kimi-K3	Qwen3.8-27B
Our harness	30.58 ± 2.00	22.77 ± 4.27	24.62 ± 4.02	6.42 ± 1.19
Our harness, w/o perception	30.29 ± 4.92	10.95 ± 2.80	8.71 ± 2.91	1.03 ± 0.30
Minimal harness (ARC-AGI-3 alike)	11.33 ± 3.02	3.31 ± 1.11	0.00 ± 0.00	0.01 ± 0.03
OSS harness: Prime-Agent	68.11 ± 6.33*	59.88 ± 0.75*	47.99 ± 3.96	26.60 ± 6.39
model-attended solving	55.29 ± 6.68	37.24 ± 4.66	42.22 ± 4.53	23.53 ± 3.87
verbal-only solving	4.37 ± 1.45	0.14 ± 0.05	9.20 ± 4.38	5.86 ± 3.90

3.2 Observation and Harness Comparisons

Table 1 shows scores of a model together with a harness, so we measure how much they depend on the harness in two directions, removing components from it and adding tools beyond it in Table 2.

Removing harness components. Replacing the ASCII board with the raw 64×64 grid changes only the observation. RHAE-L5 then falls from 30.58 to 30.29 for Opus-5, from 22.77 to 10.95 for Opus-4.8, from 24.62 to 8.71 for Kimi-K3, and from 6.42 to 1.03 for Qwen3.8-27B. The size of the drop differs widely across models, and the ordering of the mean scores of Opus-4.8 and Kimi-K3 reverses between the two settings, so the observation representation affects both the scores and the observed comparisons between models. The minimal harness goes further and removes the observation, the memory, and the interaction support at once. Under it, RHAE-L5 is 11.33 for Opus-5, 3.31 for Opus-4.8, and at most 0.01 for the other two models. Because several components are removed together, this result shows the combined effect of multiple harness components.

Table 3: **Ground-truth rule cards as a prompt-time probe**. A ground-truth rule card is a hand-written natural-language statement of a game's true rules (win condition and mechanics, transcribed from the game engine), which we supply in the system prompt at evaluation time only, as an oracle probe that separates a policy's failure to discover the rules from its failure to act on them. Eight held-out validation games, 5 evaluation seeds, 8192-token output cap, thinking off.

Policy	Extra prompt	RHAE-L5	ΔRHAE-L5	Games cleared	Efficiency	Plan-less calls
Qwen3.8-27B	none	8.0 ± 5.0	–	5/40	0.39	1%
	generic text	9.1 ± 2.5	+1.1	6/40	0.44	0%
	GT rule card	10.1 ± 2.7	+2.1	6/40	0.48	1%
Opus 5 (frontier)	none	59.9 ± 11.4	–	30/40	0.74	3%
	generic text	63.6 ± 3.7	+3.7	34/40	0.72	4%
	GT rule card	97.8 ± 0.7	+37.8	40/40	0.99	0%

Expanded tool setting. Prime-Agent adds code execution around the model, so it evaluates the same games under an expanded tool setting, and the comparison measures performance under different agent configurations rather than isolating the effect of code execution alone. It achieves higher scores for all four models (68.11, 59.88, 47.99, and 26.60 on RHAE-L5), showing that measured performance depends strongly on the surrounding agent configuration. To audit how the levels were solved, we rescore the same runs post hoc by trajectory behaviour: one rescoring excludes levels on which code submitted three or more candidate paths in one turn and let the engine verdict select the answer, and the other keeps only levels whose winning moves the model produced without code. With typed moves only, the scores fall to 4.37, 0.14, 9.20, and 5.86, so most credited levels involved programmatic search or solution generation. These rescored values describe the runs as they happened and are not estimates of what the models would score without code, because later levels may build on earlier code-based solutions and code that fits candidate rules may itself take part in hypothesis construction. More details and example trajectories are given in Appendix B.5.

3.3 Effect of Providing Ground-Truth Rules

Providing ground-truth rules probes how much performance improves when the model no longer needs to infer the stated game mechanics from interaction. For each of the eight validation games we write a rule card, a natural-language statement of its win condition and mechanics transcribed from the engine, including the action mapping, and place it in the system prompt at evaluation time. As a control, a generic-text prompt of matched length gives the same action mapping together with general guidance on observation, hypothesis testing, and efficiency, but states no rule of the game. Both prompts include the same action mapping and are matched in length, while differing in their task-specific content; details in Appendix B.1. The probe uses 5 seeds, thinking off, an 8,192-token output cap, and the first five levels of each game, a configuration different from Table 1, so scores are not comparable across tables. Table 3 reports Opus-5 and the Qwen3.8-27B starting checkpoint; Appendix B.1 reports the full probe, including the RL-trained checkpoints discussed in Section 4.

The two models respond to the rule card very differently. Relative to the length-matched generic prompt, the rule card raises Opus-5 by 34.1 points to 97.8 RHAE-L5 with every evaluated level solved in all 40 game runs, so explicit rule access is sufficient to close the gap to the ceiling in this probe; this indicates that acquiring the rules is a major difficulty for this model on these games. For Qwen3.8-27B, the rule card yields an observed mean difference of 1.0 points over the generic prompt, to 10.1, with 6 of 40 game runs clearing all evaluated levels in both conditions. Substantial difficulty therefore still remains for this model when the rules are provided, indicating limitations beyond acquiring the rule description. These results for the two models describe them as they are; whether RL on games with hidden rules improves performance on rules held out from training, and whether any improvement extends beyond the suite, is the subject of Section 4.

4 RL Training: Does It Transfer beyond Training Rules

This section addresses the second question of the introduction, whether RL on games with hidden rules improves performance on rules held out from training. We describe the training setup, report learning and transfer within the suite on the validation and private test sets, compare credit-assignment schemes, and report transfer to external benchmarks.

Training setup. We train Qwen3.5-9B and Qwen3.8-27B on the 22 training games of Section 2.2 with the Miles framework (Team, 2026). Each training episode plays one game from its first level

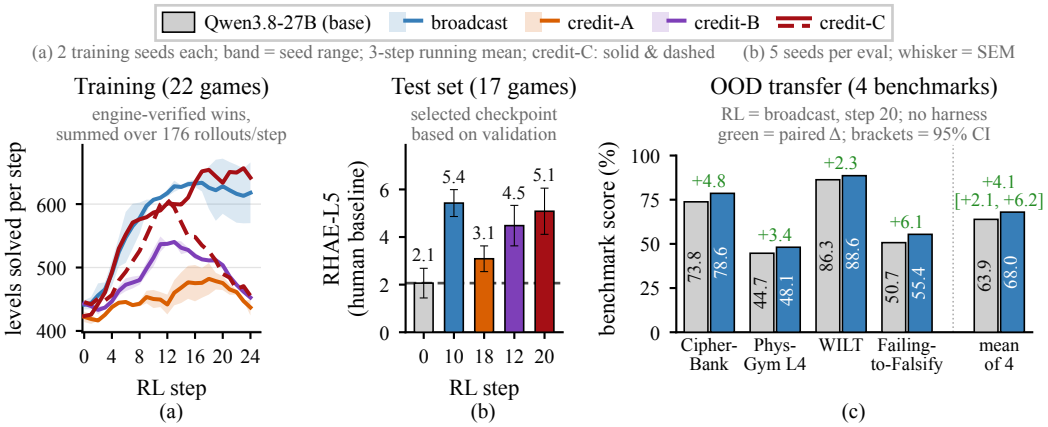


Figure 3: (a) RL training dynamics; and (b,c) generalization to test set (unseen rules) and external discovery tasks.

under the harness of Section 2.4, using game levels capped at first ten, and ends when all of them are solved, after 300 LLM calls, or when a level remains unsolved after its 300-action budget. The reward is outcome-based: $+1$ per solved level, $+0.5$ for solving all levels of the game, and -0.005 per LLM call to encourage action efficiency. The baseline algorithm is GRPO (Shao et al., 2024) with group centering by game, batch-level whitening, and asymmetric clipping, and it broadcasts each episode's advantage to every LLM call of that episode; Appendix B.3 gives the objective and hyperparameters. Each iteration samples 8 episodes for each games, and training runs for 30 iterations. Thinking is disabled during training so that the model's reasoning goes into the `<meta>` field required by the harness and rollouts stay short; all numbers in this section are therefore evaluated with thinking off and are not comparable to Table 1. Checkpoints are selected on the 8 validation games with 3 seeds, and the selected checkpoint is evaluated on the 17 private test games with 5 seeds.

Learning and transfer within the suite. Figure 6 in Appendix B.3 shows training reward and validation scores for both models under identical settings. For Qwen3.8-27B, training reward rises over iterations, and validation RHAE-L5 rises on both held-out compositions and held-out primitives. The two validation groups start at different levels, with held-out primitives near zero, and reach similar values by the end of training. For Qwen3.5-9B, neither training reward nor validation score improves under the same settings. On the private test set, the selected Qwen3.8-27B checkpoint raises RHAE-L5 from 2.1 to 5.4 and the number of solved level slots from 67 to 122 out of 1,700. These gains are measured on rules absent from training, so they reflect transfer beyond the training games; what the model has learned that transfers is examined on the external benchmarks below.

Credit assignment. Outcome-only reward leaves open how credit is distributed over the LLM calls of an episode, and we compare four schemes on Qwen3.8-27B. Broadcast assigns the episode advantage to every call. Credit-A groups calls by attempt within each level and computes credit per group, which makes the signal sparser. Credit-B increases the weight of the last successful attempt within each level. Credit-C additionally increases the weight of earlier attempts that proposed a rule hypothesis surviving to the end of a successful episode. Figure 3 and Appendix B.3 show that within the family, credit-A is below credit-B, which is below credit-C, and that credit-C is on par with broadcast. The two credit-C seeds diverge early in training, which we attribute to the noisiness of survival as a proxy for hypothesis correctness: a hypothesis can survive an episode without being correct, and the resulting credit is then misdirected. We therefore use broadcast for the remaining experiments. The behavioral study along the RL training is shown in Figure 7 (in Appendix B.3).

Transfer to external benchmarks. To test whether the gains extend beyond grid puzzles, we evaluate the base model and the RL model (broadcast) on four external benchmarks of rule inference and hypothesis testing, all with thinking on and a 32k-token cap. CipherBank (Li et al., 2025) measures exact-match accuracy on the 786 items of its author-invented ciphers. PhysGym (Chen et al., 2026b) measures the success rate on its 97 level-4 problems, which give the least prior description of the physical system. WILT (Banatt et al., 2024) measures elim@5, the fraction of predefined hypotheses eliminated within the first five experiments. Failing-to-Falsify (Jhaveri et al., 2026) measures the incompatibility fraction between a model's stated hypothesis and the tests it chooses, where a higher value indicates more falsification-seeking behavior. At iteration 20, the broadcast checkpoint improves

Table 4: Comparison of related benchmarks. Hidden rules indicates interactive inference of rules not supplied to the agent. Split identifies the unit held out between training and evaluation in the reported experiments; evaluation only indicates that no training experiment is reported for evaluated models.

Benchmark	Hidden rules	Observation	Training–evaluation split
ARC-AGI-3 (ARC Prize Foundation, 2026a)	✓	Pixel grid	Evaluation only
DiG-Bench (Battleday et al., 2026)	✓	1D string	Evaluation only
MazeBench (Pappas & Pappas, 2026)	✗	3D render (also ASCII/JSON)	Evaluation only
gg-bench (Verma et al., 2025)	✗	Text	Evaluation only (LLMs)
lmgame-Bench (Hu et al., 2026)	✗	Frames or text	By game
Enigmata (Chen et al., 2026a)	✗	Text	By instance
HardcoreLogic (Liang et al., 2026)	✗	Text	Evaluation only
WITNESS (ours)	✓	2D ASCII board	By rule (validation): compositions, primitives

the mean over the four benchmarks by 4.1 points (95% CI 2.1 to 6.2), with per-benchmark intervals excluding zero for CipherBank (+4.8) and WILT (+2.3), a lower bound at zero for PhysGym (+3.4), and an interval including zero for Failing-to-Falsify (+6.1). These results indicate that part of what is learned on WitnessGym transfers to rule inference outside the suite, with the clearest gains on cipher decryption and experiment design. Table 7 in Appendix B.4 reports more detailed scores.

5 Related Work

Interactive rule discovery and puzzle benchmarks. Puzzle and game benchmarks differ in what the agent observes and in how training and evaluation are divided; Table 4 compares these properties. ARC-AGI-3 (ARC Prize Foundation, 2026a) presents interactive games as integer grid frames whose rules and goals are not given, so a score reflects both the interpretation of the frames and the discovery of the rules. Other listed benchmarks avoid visual recognition such as DiG-Bench (Battleday et al., 2026) and MazeBench (Pappas & Pappas, 2026). Some benchmarks (Verma et al., 2025; Chen et al., 2026a;c; Li et al., 2026) state the rules in the prompt, whereas lmgame-Bench (Hu et al., 2026) uses published game titles whose mechanics may have appeared in pretraining data. HardcoreLogic (Liang et al., 2026) transforms familiar puzzle families into long-tail variants and reports large performance drops on the variants. Relative to these designs, WITNESS exposes an entity-level two-dimensional board, hides the rules, and organizes validation games by their relationship to the RL training rules. This combination supports controlled observation and rule-access probes, together with evaluation of transfer across specified rule splits.

Learning and generalization through games. Beyond evaluation, games serve as RL environments, with the engine providing verifiable outcome rewards and levels serving as tasks. Training on games has been reported to transfer beyond the training games in several settings. Tong et al. (2026) and Wang et al. (2026) train vision-language models on game-derived tasks and report gains on multimodal benchmarks. Hu et al. (2026) report that RL on Sokoban and Tetris improves performance on other games in the suite and on the planning tasks Blocksworld and WebShop. Liu et al. (2026) report gains of up to 10 points on eight reasoning benchmarks from self-play on zero-sum games, and Verma et al. (2025) train RL agents by self-play on generated games. We study whether training language models on games with hidden rules improves performance on held-out compositions and held-out primitives, evaluating these two forms of transfer separately.

We defer broader related work discussion on hypothesis testing, scientific discovery, and agent harnesses in Appendix A.

6 Conclusion

We introduced WITNESS, a grid-based environment for studying interactive rule discovery, comprising WitnessBench for evaluation, WitnessGym for RL training, and WitnessForge for game generation. With this environment we asked what limits current language models and whether RL improves performance. On the first question, the highest-scoring model on RHAE-L5 under a shared harness solves only 24% of private test level slots. On the second question, RL training raises Qwen3.8-27B's private test RHAE-L5 from 2.1 to 5.4, improves performance on held-out compositions and held-out primitives, and yields an average gain of 4.1 percentage points across four external benchmarks, providing evidence of transfer beyond the training rules. We hope this environment, composed of the challenging benchmark and generalizable RL training task, can foster further research in improving genuine reasoning intelligence of AI particularly in unfamiliar environments.

AI USE STATEMENT

We used generative AI tools (e.g., Opus-5, Qwen) as part of the experimental methodology of this work, specifically as the subjects of our rule-discovery evaluations, and within an agentic pipeline to generate WITNESS game prompts, novel rules, games, and environments. We also acknowledge the use of language models for writing assistance and language editing. The core research methodology, experimental design, analysis, and scientific conclusions were developed and verified by the authors. All AI-assisted code, text, and experimental artifacts were reviewed by the authors, who take full responsibility for the final content of this work.

ETHICS STATEMENT

This work focuses on developing and evaluating language models in abstract, synthetic environments (WITNESS). Our datasets and environments are entirely procedurally generated and do not contain personally identifiable information (PII), human subject data, or societal biases.

Broadly speaking, "automated science" is a capability that, if fully realized in domains like chemistry or biology, could carry dual-use risks regarding the automated discovery of harmful materials. While our long-term motivation involves advancing agents for "automated science", the current research is strictly confined to highly abstract, 2D grid-based logic puzzles. Our agent operates in a sandbox without access to external real-world databases, mitigating direct risks of malicious use.

In addition, while extensive evaluating and RL training of LLMs (up to 27B parameters) carry an environmental cost due to compute requirements, we mitigate this by releasing our benchmark (WitnessBench) and RL training suite (WitnessGym). This open release allows the community to reproduce our results and build upon our work without incurring the redundant carbon footprint.

REPRODUCIBILITY STATEMENT

We are committed to ensuring that all evaluations on validation games and training results introduced in this paper are fully reproducible. To achieve this, we have taken the following steps:

- **Code and Environments:** The source code for the WITNESS environment engine and the evaluation harness are included in the supplementary materials.
- **Model Evaluation:** Exact prompt templates and evaluation settings (e.g., model version identifiers and sampling parameters) for the 18 models are detailed in Appendix B.6 and B.7. We provide a link to an anonymous webpage with playable demos and visualized traces from frontier models.
- **RL Training Details:** For the RL experiments on the Qwen3.8-27B model, details such as training hyperparameters and hardware specifications (compute resources and estimated runtime) are provided in Appendix B.3. The training scripts are also included in the supplementary materials.

Upon de-anonymization, all code, training and validation datasets, and model weights will be publicly released under an open-source license. To prevent data contamination, the games in the private test set and their corresponding model trajectories are deliberately withheld from the supplementary material.

REFERENCES

ARC Prize Foundation. Arc-agi-3: Interactive abstraction and reasoning corpus, 2026a. `https://arcprize.org/arc-agi/3/`.

ARC Prize Foundation. Arc-agi-3: A new challenge for frontier agentic intelligence. Technical report, ARC Prize Foundation, 2026b.

Eryk Banatt, Jonathan Cheng, Skanda Vaidyanath, and Tiffany Hwu. Wilt: A multi-turn, memorization-robust inductive logic benchmark for llms. *arXiv preprint arXiv:2410.10998*, 2024.

Ruairidh M Battleday, Kai Sandbrink, Jimi Cullen-Drohan, Zihan Yan, Timothy Muller, Clare Maguire, Ales Kubicek, Fraser Greenlee-Scott, Sukrit Sumant, Tri Dao, et al. Dig-bench: Discovery in games. *arXiv preprint arXiv:2608.12593*, 2026.

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. *NeurIPS*, 38:3613–3661, 2026a.

Yimeng Chen, Piotr Piekos, Mateusz Ostaszewski, Firas Laakom, and Jürgen Schmidhuber. Physgym: Benchmarking llms in interactive physics discovery with controlled priors. *NeurIPS*, 38, 2026b.

Zhenhao Chen, Yongqiang Chen, Chenxi Liu, Junchi Yu, Xiangchen Song, Zijian Li, Jialin Li, Philip Torr, Bo Han, and Kun Zhang. Causalgame: Benchmarking causal thinking of llm agents in games. *ICML*, 2026c.

Claire Cook, Noah D Goodman, and Laura E Schulz. Where science starts: Spontaneous experiments in preschoolers' exploratory play. *Cognition*, 120(3):341–349, 2011.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

ARC Prize Foundation. Arc-agi toolkit, 2026. URL `https://docs.arcprize.org/toolkit`.

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? In *ICLR*, volume 2026, pp. 81308–81356, 2026.

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

Ayush Rajesh Jhaveri, Anthony GX-Chen, Ilia Sucholutsky, and Eunsol Choi. Failing to falsify: Evaluating and mitigating confirmation bias in language models. *arXiv preprint arXiv:2604.02485*, 2026.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, and Sami Jaghouar. Prime agent: A self-improving rlm harness. *arXiv preprint arXiv:2608.23552*, 2026.

Ross D King, Jem Rowland, Stephen G Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N Soldatova, et al. The automation of science. *Science*, 324(5923):85–89, 2009.

Hengzhi Li, Justin Zhang, Brendon Jiang, Alexander Naehu, Regan Song, Megan Tjandrasuwita, Chanakya Ekbote, Steven-Shine Chen, Adithya Balachandran, Wei Dai, et al. Puzzleworld: A benchmark for multimodal, open-ended reasoning in puzzlehunts. In *ICLR*, volume 2026, pp. 11819–11843, 2026.

Yu Li, Qizhi Pei, Mengyuan Sun, Honglin Lin, Chenlin Ming, Xin Gao, Jiang Wu, Conghui He, and Lijun Wu. Cipherbank: Exploring the boundary of llm reasoning capabilities through cryptography challenge. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5929–5965, 2025.

Jingcong Liang, Shijun Wan, Xuehai Wu, Yitong Li, Qianglong Chen, Duyu Tang, Siyuan Wang, et al. Hardcorelogic: Challenging large reasoning models with long-tail logic puzzle games. In *ICLR*, volume 2026, pp. 80011–80050, 2026.

Bo Liu, Simon Yu, Zichen Liu, Leon Guertler, 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. In *ICLR*, volume 2026, pp. 35407–35434, 2026.

Jonathan Pappas and David Pappas. Maze Bench: Visual spatial reasoning in a 3d open world, 2026. URL `https://mazebench.com`. Project oversight by Florian Brand and Sebastian Müller.

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.

Miles Team. Miles: Enterprise-grade reinforcement learning for large-scale model post-training. https://github.com/radixark/miles, 2026.

Thekla, Inc. The witness, 2016. URL http://the-witness.net/. Video game.

Jingqi Tong, Jixin Tang, Hangcheng Li, Yurong Mou, Ming Zhang, Jun Zhao, Yanbo Wen, Fan Song, Jiahao Zhan, Yuyang Lu, et al. Game-rl: Synthesizing multimodal verifiable game data to boost vlms' general reasoning. In *ICLR*, volume 2026, 2026.

Vivek Verma, David Huang, William Chen, Dan Klein, and Nicholas Tomlin. Measuring general intelligence with generated games. *arXiv preprint arXiv:2505.07215*, 2025.

Qinsi Wang, Bo Liu, Tianyi Zhou, Jing Shi, Yueqian Lin, Yiran Chen, Hai Helen Li, Kun Wan, and Wentian Zhao. Vision-zero: Scalable vlm self-improvement via strategic gamified self-play. *ICLR*, 2026.

Guanning Zeng, Jiani Wang, Wenjie Ma, Shaofeng Yin, Chenyang Wang, Shichen Liu, Angjoo Kanazawa, Wode Ni, Xiuyu Li, Andrea Zanette, and Haiwen Feng. [schema]: Frontier Models with the Right Harness Achieve ∼99% on ARC-AGI-3 Public. Impossible Research. https://schema-harness.github.io/, 2026. URL https://schema-harness.github.io/.

Table 5: Ground-truth (GT) rule cards as extra prompts for test-time ablation. A ground-truth rule card is a hand-written natural-language statement of a game's true rules (win condition and mechanics, transcribed from the game engine), which we supply in the system prompt at evaluation time only, as an oracle probe that separates a policy's failure to discover the rules from its failure to act on them. Eight held-out validation games, 5 evaluation seeds, 8192-token output cap, thinking off. A rule card does not bring great improvements on the base model, while the same card improves the RL policies significantly. Frontier models such as Opus-5 are mainly bottle-necked by rule discovery. These hint-assisted scores are only for diagnostic purpose and excluded from all other results.

Policy	Extra prompt	RHAE-L5	ΔRHAE-L5	RHAE-uncap	Games cleared	Efficiency	Plan-less calls
Base (no RL)	none	8.0 ± 5.0	–	9.9 ± 6.7	5/40	0.39	1%
	generic text	9.1 ± 2.5	+1.1	11.4 ± 5.6	6/40	0.44	0%
	GT rule card	10.1 ± 2.7	+2.1	14.8 ± 7.4	6/40	0.48	1%
Broadcast RL, step 10	none	15.9 ± 3.4	–	19.8 ± 2.1	7/40	0.50	2%
	generic text	20.2 ± 4.8	+4.3	22.3 ± 6.0	8/40	0.63	1%
	GT rule card	47.7 ± 5.0	+31.8	82.6 ± 14.8	31/40	0.69	57%
Credit-C RL, step 20	none	13.1 ± 3.5	–	15.1 ± 3.5	7/40	0.50	12%
	generic text	22.3 ± 7.8	+9.2	25.8 ± 8.3	9/40	0.61	12%
	GT rule card	48.4 ± 5.3	+35.3	91.4 ± 25.7	29/40	0.71	71%
Opus 5 (frontier)	none	59.9 ± 11.4	–	152.8 ± 26.2	30/40	0.74	3%
	generic text	63.6 ± 3.7	+3.7	195.9 ± 25.5	34/40	0.72	4%
	GT rule card	97.8 ± 0.7	+37.8	356.2 ± 13.2	40/40	0.99	0%
GPT-6 Astra (frontier)	none	63.7 ± 2.4	–	194.2 ± 14.2	25/40	0.87	2%
	generic text	68.8 ± 3.0	+5.1	245.3 ± 22.6	30/40	0.81	1%
	GT rule card	98.6 ± 0.0	+34.9	366.0 ± 1.3	40/40	0.99	0%

A Broader Related Work

Hypothesis testing and scientific discovery. The same loop of proposing a hypothesis, choosing an experiment, and revising the hypothesis from the outcome is studied under the framing of scientific discovery (King et al., 2009; Cook et al., 2011), and Fawzi et al. (2022) study how quickly models discover rules relative to their interaction budget. We adopt four evaluations from this line as external transfer targets. PhysGym (Chen et al., 2026b) evaluates the discovery of physical laws through interactive experiments, and WILT (Banatt et al., 2024) evaluates inductive rule inference from queried examples. Jhaveri et al. (2026) evaluate whether models seek evidence that would falsify their own hypotheses, and CipherBank (Li et al., 2025) evaluates the inference of hidden encodings. These evaluations probe related aspects of rule inference and hypothesis testing across domains and representations, allowing us to assess transfer beyond grid puzzles.

Agent interfaces and harnesses. The interface between the model and the environment affects what an evaluation measures. Hu et al. (2026) report that adding perception and memory modules raises the fraction of runs beating a random baseline from 60% to 86.7%. ARC-AGI-3 results likewise depend on the agent scaffold around the model, and reported scores combine the model with scaffolds that differ across submissions (ARC Prize Foundation, 2026b). Frameworks that provide search or code execution, such as Prime-Agent (Karten et al., 2026) and [schema] (Zeng et al., 2026), introduce additional computational tools and interaction policies, so their scores are not directly comparable to those obtained under a fixed action-only interface. A shared harness therefore controls one source of variation in cross-model comparisons. Section 3 evaluates all models under one harness, ablates its components to measure their effect, and reports results under such a framework as a reference point.

B Ablations, Experiment Details, Qualitative Studies, Full Prompts, and Pipeline Pseudo-code

B.1 Ablation: Given Ground-Truth Rules

To minimize the possibility of leakage of GT rules and contamination of the test set, we perform this ablation with the validation set. For a frontier model such as Opus-5, with GT rules, it is able to solve all games and levels, and highly efficiently. Given GT rules, RL-trained models tend to over-think with the rules and use up the context tokens before writing a plan. The rate of plan-less calls rises

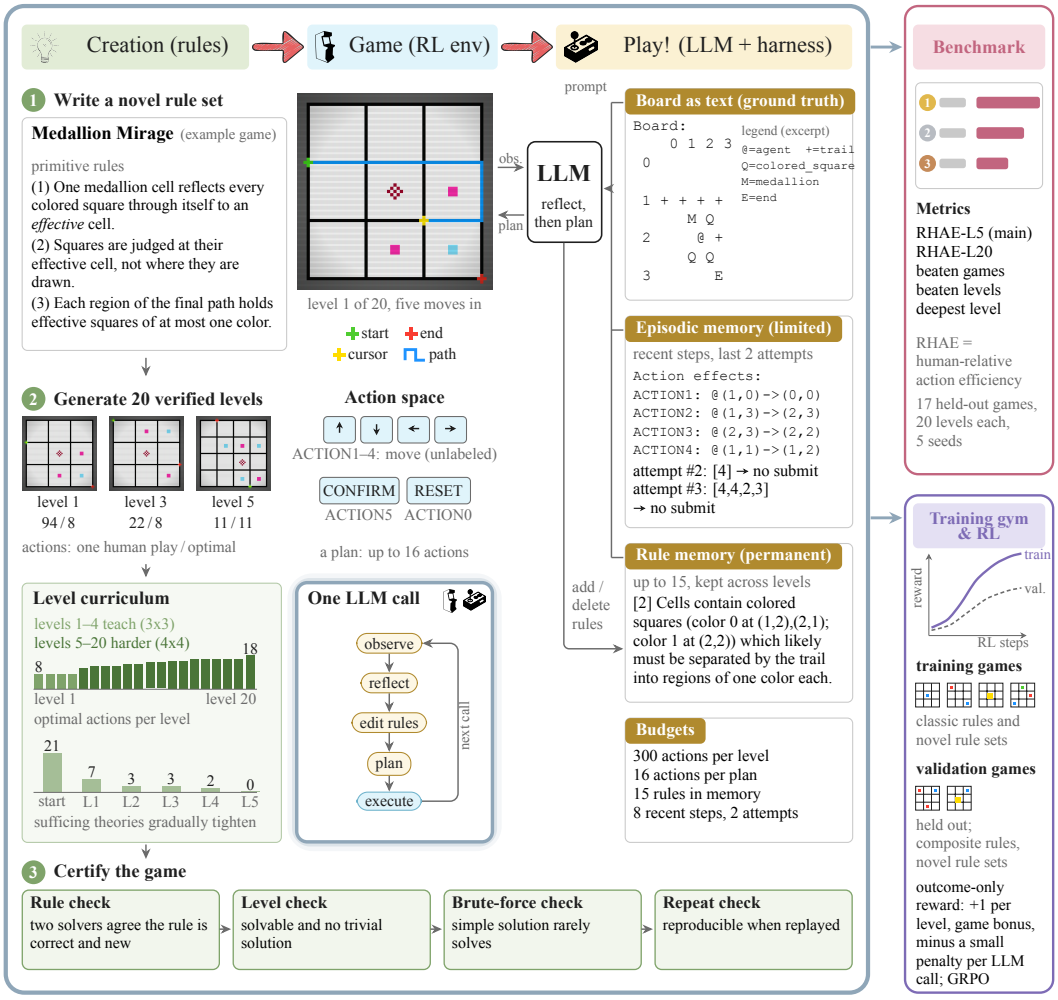


Figure 4: Cheatsheet for **W**itness-**I**nspired **T**estbed of **N**ovel **E**nvironments with **S**ymbolic **S**tructure (WITNESS).

dramatically (e.g., from 2% to 57%), especially for more complex levels, although RL-trained models also achieve a much higher RHAE score (e.g., over +31% improvement) with GT rule cards.

Generic-text control: additional text injected into the official eval harness as text content ablation for the GT rule card. It answers the question of whether injecting additional guidance that does not reveal hidden rules is also helpful. Length-matched to the GT rule cards.

The following is general guidance (given and correct). It states no rules of this game; the instruction elsewhere to rely only on observation still applies to the game itself.

Action ids. In this family of games ACTION1 = UP, ACTION2 = DOWN, ACTION3 = LEFT, ACTION4 = RIGHT, ACTION5 = CONFIRM (submits your work and moves nothing) and ACTION0 = RESET (restarts the current level; what you have learned is kept). Coordinates in the board text are written (row, col), 0-indexed, with (0,0) at the top-left.

Observation. Read the whole board text before acting: the legend, the Start and End lines, the Cell content list and the Path so far line describe the same board from different angles, so cross-check them against each other. Treat every symbol on the board as potentially meaningful until evidence shows otherwise, and note where symbols sit relative to each other and to the start and end. Prefer deliberate single-purpose tests over long speculative sequences; after each test compare the new board text with the previous one and record exactly what changed.

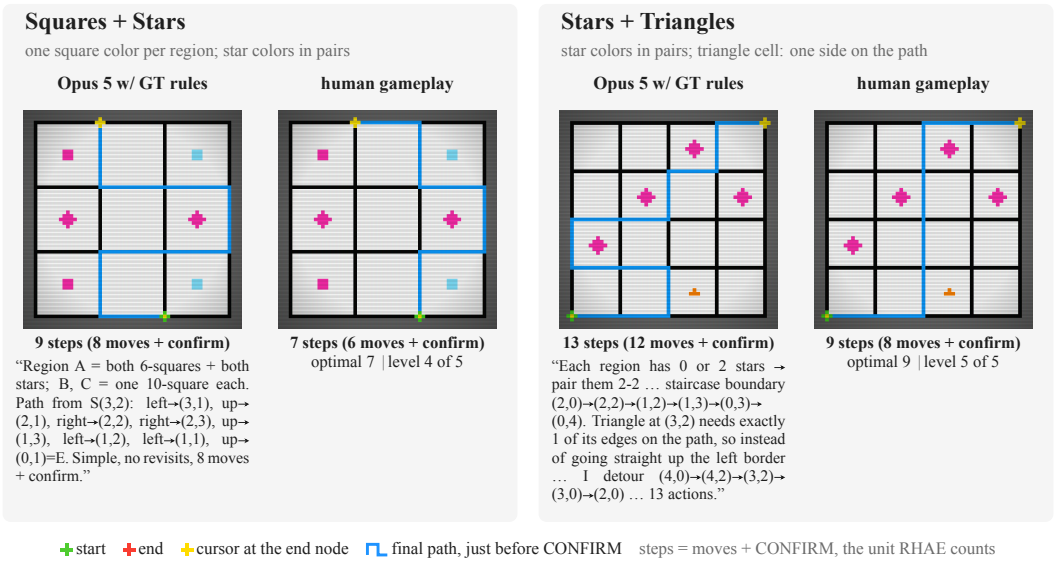


Figure 5: Why Opus-5 does not achieve perfect RHAE score (100) given GT rules: rare cases where human players complete a level more efficiently than Opus. In these levels, a human player selected a more efficient path, possibly unconsciously as it is a straight-forward path; the opus model selected non-optimal paths in 3/5 and 1/5 seeds, respectively. For most other cases, especially in earlier levels where a human player performed numerous steps to discover rules, Opus given GT rules executes much more efficiently, achieving RHAE-uncap scores that far exceed 100.

Rules. Keep rules short, testable and one sentence each; delete a rule as soon as an observation contradicts it instead of patching it with exceptions. Distinguish what you observed from what you inferred, and mark inferences as tentative until a second, independent observation supports them. If several rules could explain the same evidence, prefer the simplest one that explains every observation, and design the next action to tell the candidates apart.

Efficiency. Every action counts toward the step budget and the efficiency score, including moves that do nothing, failed confirms and resets; a level scores higher when it is solved in fewer actions, so think before acting and stop probing at random once the mechanics are clear.

B.2 Ablation: Level Diversity vs Rule Diversity

We trained with games of fixed levels. It is essential to observe whether we should add level diversity during training, i.e., training with the same games (identical rules) but different level sets (specific boards/tasks vary). In Table 6, it shows that the gains in trained layouts carry over to the unseen layouts. Therefore, level diversity is not necessary and we can focus on rule diversity.

Table 6: Transfer of RL gains to unseen layouts of the same rules. For each training game that has an alternate level set we evaluate checkpoints on the trained set and on a reserved set that was never used in training (same rules, different boards; the two layouts differ only in levels). Cells: levels completed / action efficiency on completed levels / official game score. The gains on trained layouts carry over to the unseen layouts; the gap between trained and unseen does not grow with training.

Checkpoint	Layout	ft10 (Δslots 4–5)	ft14 (Δslots 1–2)	ft15 (Δslots 0–4, 7)	ft16 (Δslots 4–7)
Base (no RL)	trained	1.19±0.32 / 0.29 / 0.6	2.25±0.37 / 0.46 / 3.0	2.81±0.32 / 0.45 / 3.4	2.06±0.46 / 0.59 / 4.5
	unseen	0.75±0.37 / 0.38 / 1.1	1.31±0.38 / 0.41 / 1.9	2.12±0.38 / 0.43 / 2.7	3.69±0.38 / 0.40 / 4.5
RL, step 10	trained	6.44±0.80 / 0.67 / 40.0	3.12±0.40 / 0.50 / 5.8	2.50±0.48 / 0.59 / 4.6	4.31±0.30 / 0.48 / 8.2
	unseen	5.94±0.63 / 0.64 / 25.2	2.88±0.48 / 0.41 / 4.9	2.62±0.38 / 0.44 / 3.9	4.62±0.34 / 0.62 / 13.7
RL, step 30	trained	3.75±0.81 / 0.52 / 12.9	3.44±0.27 / 0.51 / 7.0	1.88±0.32 / 0.58 / 2.7	2.44±0.46 / 0.58 / 5.3
	unseen	5.00±0.80 / 0.53 / 21.1	2.12±0.41 / 0.31 / 2.8	2.31±0.38 / 0.38 / 2.4	3.25±0.50 / 0.59 / 8.5

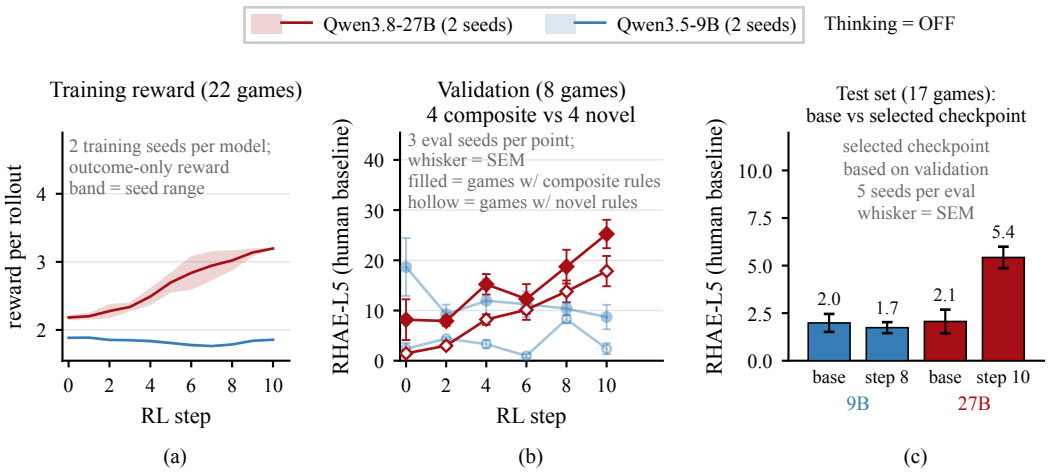


Figure 6: Comparing models of different sizes (9B vs 27B) with identical training settings. (a) 9B model does not experience useful learning signals while 27B learns to beat more levels. These findings corroborate the belief that complex behaviors emerge during model scaling; (b) the validation performance on composite of known primitive rules and unseen rules starts differently but converge. It suggests what model learns during RL training is generalized capability rather than memorizing specific patterns; (c) training on games of primitive rules generalized to test games of unseen rules.

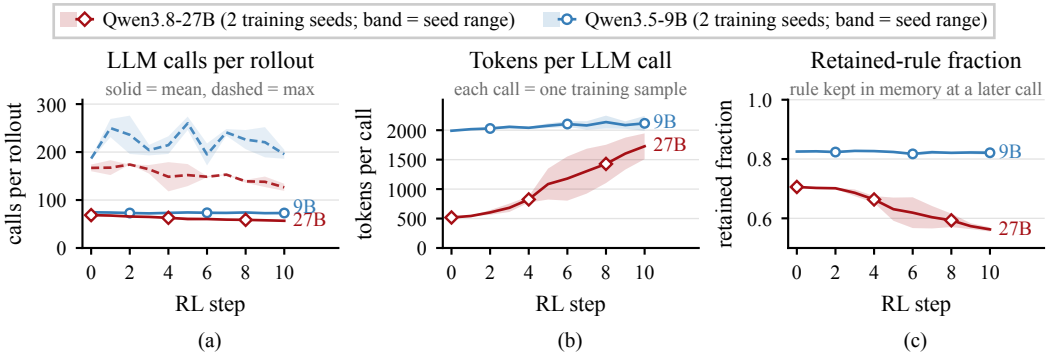


Figure 7: Behavior comparison between 9B and 27B models. (a) In contrast to 9B model, 27B model learns to complete more levels with less LLM calls within a game rollout as the training progresses; (b) 27B model learns to reflect more within each LLM call to plan high-quality actions, while 9B model consistently outputs verbose tokens that do not improve level-solving; (c) 27B model learns to preserve smaller fraction of rules it proposed earlier, while 9B model keeps generating and preserving most low-level rule hypotheses that do not contribute to puzzle solving.

B.3 Detailed RL Settings and Training Dynamics

Settings. We train Qwen3.5-9B and Qwen3.8-27B using Miles Team (2026) on B300 GPUs. At each training iteration, we sample 8 independent episodes for each of the 22 training games, giving 176 episodes before filtering. All episodes use the same set of levels during training and start from the first level. An episode ends when the agent completes all ten training levels (or all five, for the two training games that have only five levels), reaches 300 LLM calls, or exhausts the 300-action budget for an unsolved level.

At each call, the model receives the current ASCII board with its rule memory and recent attempts. It returns an analysis, memory operations, and an optional plan of up to 16 actions. The harness executes the plan before querying the model again. We sample at temperature 1 without specified

top-p or top-k filtering and allow up to 8,192 response tokens per call. Native thinking mode is disabled. The model writes its analysis in the `<meta>` field required by the harness.

Under our 27B model training settings, it takes about 119 hours of wall-clock time on one $8 \times B300$ node ($\approx$ 950 GPU-hours) for the full 30-step training, with each step spending an average of 4.0 hours. Because training is synchronous and the trainer and inference engine are co-located on the same GPUs, sampling and training proceed alternately; the sampling stage, which waits for the slowest of the 176 rollouts, consumes about 75% of total wall-clock.

Reward. The agent receives a reward of 1 for each completed level and an additional 0.5 bonus for completing all ten levels (the two five-level games cannot earn this bonus). Each LLM call has a cost of 0.005. For an episode τ, the return is

$$R(\tau) = n_{\text{solved}}(\tau) + 0.5\,\mathbb{1}[n_{\text{solved}}(\tau) = 10] - 0.005\,S(\tau), \tag{3}$$

where $S(\tau)$ counts all LLM calls in the episode, including calls whose responses are truncated or whose plans fail to advance the board. Level completion is checked by the game engine. We do not use a separate reward for the correctness of the model's explanations or proposed rules.

Rollout groups and advantage estimation. We group the sampled episodes by training game. Episodes interrupted by infrastructure failures are removed before computing advantages. Episodes that exhaust an action or LLM-call budget are retained. Let $\mathcal{B}$ denote the retained rollout batch and $\mathcal{G}_g$ its group of episodes for game g. Each sampled episode is counted separately, including episodes with identical trajectories.

For each nonempty group, we use its mean return as the baseline:

$$\widetilde{A}(\tau) = R(\tau) - \frac{1}{|\mathcal{G}_g|} \sum_{\tau' \in \mathcal{G}_g} R(\tau'), \qquad \tau \in \mathcal{G}_g. \tag{4}$$

The advantages are centered and whitened once across the nonempty retained batch as following:

$$\begin{aligned} \mu_A &= \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \widetilde{A}(\tau), \\ \sigma_A^2 &= \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \big(\widetilde{A}(\tau) - \mu_A\big)^2, \\ A(\tau) &= \begin{cases} \dfrac{\widetilde{A}(\tau) - \mu_A}{\sigma_A + 10^{-8}}, & \sigma_A \geq 10^{-8}, \\ 0, & \sigma_A < 10^{-8}. \end{cases} \end{aligned} \tag{5}$$

Each episode contributes once to these statistics, regardless of its number of calls or tokens. We monitor both return statistics and the fraction of groups with varying solve counts.

Broadcast baseline. Let $A_j(\tau)$ be the advantage assigned to call j in episode τ. The broadcast baseline sets $A_j(\tau) = A(\tau)$ for every call and uses this value for all its retained response tokens.

Policy objective. For call j, let x_j and y_j denote the stored prompt and response. Let $\pi_{\theta_{\text{old}}}$ be the policy used to collect the current batch. Writing $y_{j,t}$ for token t of response y_j, the importance ratio is

$$\rho_{j,t}(\theta) = \frac{\pi_\theta(y_{j,t} \mid x_j, y_{j,<t})}{\pi_{\theta_{\text{old}}}(y_{j,t} \mid x_j, y_{j,<t})}. \tag{6}$$

Here $y_{j,<t}$ denotes the response tokens preceding position t. We use the PPO clipped surrogate with lower and upper clipping thresholds of $1 - \varepsilon_{\text{lo}} = 0.8$ and $1 + \varepsilon_{\text{hi}} = 1.28$. The token loss is

$$\ell_{j,t}(\theta; \tau) = -\min\Big\{\rho_{j,t}(\theta) A_j(\tau),\ \ \text{clip}\big(\rho_{j,t}(\theta), 1 - \varepsilon_{\text{lo}}, 1 + \varepsilon_{\text{hi}}\big) A_j(\tau)\Big\}. \tag{7}$$

The old policy and advantages are held fixed during optimization. We use no KL penalty, entropy bonus, or dual clipping.

Token masking and loss aggregation. Responses that reach the output limit are truncated, but the episode continues. We exclude these responses from the policy loss. Let $m_{j,t} = 1$ for response tokens in retained calls and $m_{j,t} = 0$ for tokens in truncated calls. Prompt and environment-observation

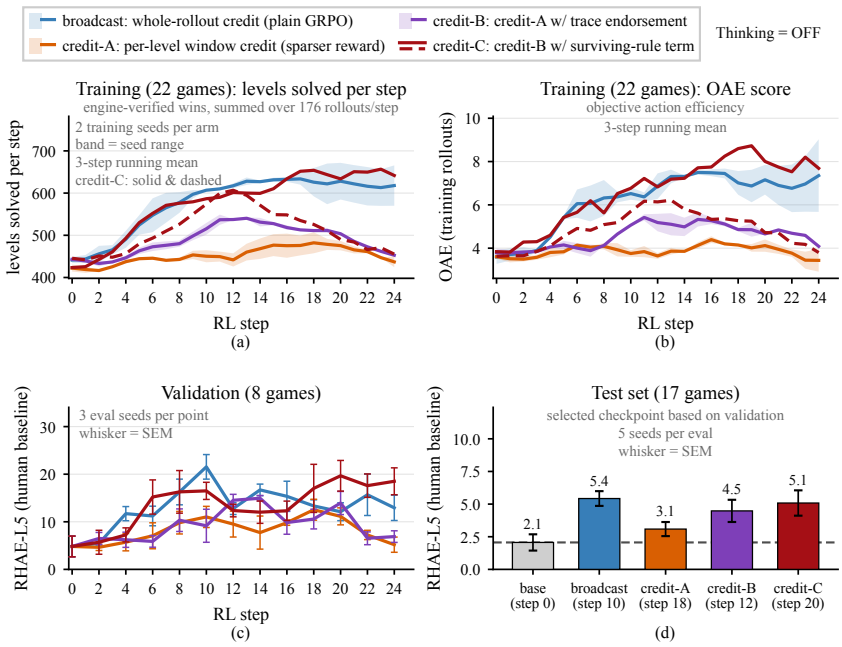


Figure 8: Comparison of RL algorithms: various credit assignment methods. Credit-C is shown with two curves, one with each seed, to exhibit its divergence during training, due to the noise in deterministically keyed rules.

tokens are not trained. Episodes contain different numbers of calls and response tokens. We first average the retained token losses within each call, then sum these per-call means over the batch, normalizing by the number of sampled episodes. For a non-empty batch in which every episode has at least one retained response token, we have

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{\tau \in \mathcal{B}} \sum_{j} \frac{\sum_t m_{j,t} \ell_{j,t}(\theta; \tau)}{\sum_t m_{j,t}}. \tag{8}$$

Here N is the trainer's fixed batch normalizer (176, the number of episodes sampled per iteration, in our configuration). The inner sum runs over the retained calls j of episode τ; both token sums run over the response tokens of call j; the episode dependence of $m_{j,t}$ and $\rho_{j,t}$ is implicit. Only retained response tokens enter the numerator and denominator. Each retained call has equal weight in the batch average, so an episode's weight is proportional to its number of retained calls. Within a call, tokens share the call's weight equally.

For the broadcast baseline mentioned above, every call uses the same episode advantage, $A_j(\tau) = A(\tau)$. Substituting this into Equation 8 gives

$$\mathcal{L}_{\text{broadcast}}(\theta) = -\frac{1}{N} \sum_{\tau \in \mathcal{B}} \sum_{j} \frac{1}{\sum_t m_{j,t}} \sum_t m_{j,t} \cdot \min\Big\{ \rho_{j,t}(\theta) A(\tau),\ \text{clip}\big(\rho_{j,t}(\theta), 0.8, 1.28\big) A(\tau) \Big\}. \tag{9}$$

Credit-assignment variants. The three variants compared in Section 4 replace the broadcast advantage with the call-specific advantages. They use the same episode sampling, episode reward defined in Equation 3, clipped token loss in Equation 7, and the call-mean loss reduction in Equation 8. They differ in how the call advantages are computed and which calls enter the loss. An advantage of the call is shared by all its retained response tokens. For the credit variants, let $T_j = \sum_t m_{j,t}$ be the number of trainable response tokens in call j.

Credit-A: per-level-window credit. For each episode τ, we partition its calls into windows according to the level recorded for each call by the game engine. Let $W_{\tau,\ell}$ contain the calls assigned to level ℓ, and let $W(j)$ be the window containing call j.

The local return used for window comparisons is

$$R(W) = \mathbb{1}[\mathrm{solved}(W)]\big(1 + 0.5\,\mathbb{1}[\mathrm{bonus}(W)]\big) - 0.005\,|W|, \tag{10}$$

where $|W|$ counts LLM calls, $\mathrm{solved}(W)$ indicates that the window solves its assigned level, and $\mathrm{bonus}(W)$ marks a window that completes the final level. This local return is distinct from the episode return, whose cost is per environment action. We use $R(W)$ for the comparison below so a call that sweeps several levels contributes only one solved-window outcome to its sibling group.

For game g and level ℓ, let $\mathcal{W}_{g,\ell}$ contain the windows from the sampled episodes assigned to that game and level. We center each window return against its siblings:

$$\widetilde{A}(W) = R(W) - \frac{1}{|\mathcal{W}_{g,\ell}|} \sum_{W' \in \mathcal{W}_{g,\ell}} R(W'), \quad W \in \mathcal{W}_{g,\ell}. \tag{11}$$

We remove all calls in a sibling group if it has fewer than two windows, every window is solved, or no window is solved. Such groups lack a contrast in level completion even if their call counts differ. Removed calls do not enter the policy loss or the normalization below. They are removed through the trainer's sample-removal hook rather than assigned zero advantage.

Let σ be the standard deviation of $\widetilde{A}(W(j))$ across calls that remain trainable after this filter, counting the value once per call with $T_j > 0$. When $\sigma > 0$, Credit-A assigns

$$A^{\mathrm{A}}_j(\tau) = \frac{\widetilde{A}\big(W(j)\big)}{\sigma}. \tag{12}$$

We divide by the batch-wide scale without subtracting the mean over calls. Sibling centering already sets the zero point for each group and a further mean over calls weights windows by their call counts.

Truncated calls remain in their windows when computing $R(W)$, which count in the window cost $0.005\,|W|$, and forming the sibling groups, including windows made entirely of truncated calls. They have $T_j = 0$ and contribute neither to the loss nor to the calculation of σ.

Credit-B: crediting the calls that solved. Credit-B uses the same windows, sibling centering, and scale σ as Credit-A. It changes call weights only in solved windows with $\widetilde{A}(W) > 0$, elsewhere $A^{\mathrm{B}}_j(\tau) = A^{\mathrm{A}}_j(\tau)$.

Within a level, an agent-chosen reset or a failed submit starts a new attempt. A failed submit is a confirm action that does not advance the level and clears the drawn path. The terminal give-up reset does not start an attempt. A call belongs to the attempt in which its plan began. If a boundary occurs during that plan, the new attempt starts with the next call.

Under Credit-B, a call in an eligible window is *credited* if it belongs to the final attempt or chooses a reset. A failed submit only creates an attempt boundary and this event alone does not credit its call. For either variant $v \in \{\mathrm{B}, \mathrm{C}\}$, a credited call receives raw weight 1 and every other call receives raw weight 0.2. Here *credited* names the higher raw-weight class; the remaining calls still have non-zero raw weight. For a window with at least one trainable token, these weights are rescaled as

$$w^{(v)}_j = \widetilde{w}^{(v)}_j \frac{\sum_{k \in W} T_k}{\sum_{k \in W} \widetilde{w}^{(v)}_k T_k}, \qquad \sum_{j \in W} w^{(v)}_j T_j = \sum_{j \in W} T_j. \tag{13}$$

Here $\widetilde{w}^{(v)}_j$ denotes the raw weight assigned under variant v. In eligible windows, the call advantage is

$$A^{(v)}_j(\tau) = w^{(v)}_j \frac{\widetilde{A}(W(j))}{\sigma}, \qquad v \in \{\mathrm{B}, \mathrm{C}\}. \tag{14}$$

In other windows, both variants use $A^{\mathrm{A}}_j(\tau)$. The raw weights have a maximum relative ratio of five, and the rescaling preserves $\sum_{j \in W} w^{(v)}_j T_j$ in each window. Under the loss above, this is a token-mass constraint and it does not in general preserve the sum of call weights or the window's contribution to the loss. We compute σ from the unweighted centered values, before applying the weights.

Below we discuss on Credit-C, which uses the same weighting rule and additionally endorses the surviving rules.

Table 7: Training dynamics of OOD generalization: credit assignment (broadcast vs. credit-C) × training step (10 vs. 20) on the four benchmarks. During RL training, both models have progressive OOD gains against the base model.

Model	Step	**CipherBank custom ciphers (%)** cipher decryption 786 items, 2 reps	**PhysGym L4 success (%)** law discovery 97 problems, 3 reps	**WILT elim@5 (%)** rule discovery 50 rules, 5 reps	**Failing-to-Falsify incompat. fraction (%)** Wason 2-4-6 80 instances, 5 reps	**Mean Δ over 4 benchmarks** [95% CI]
Qwen3.8-27B (base)	–	73.8 ± 0.2	44.7 ± 1.6	86.3 ± 1.6	50.7 ± 5.1 paired subset: 49.3–49.6	–
+ RL broadcast	10	77.6 ± 2.3 (+3.8)	45.4 ± 1.8 (+0.7)	87.8 ± 1.1 (+1.5)	51.2 ± 3.5 (+0.5)	+1.6 [−0.3, +3.6]
	20	78.6 ± 1.3 (+4.8)	48.1 ± 2.1 (+3.4)	88.6 ± 1.7 (+2.3)	55.4 ± 4.9 (+6.1)	**+4.1** [+2.1, +6.2]
+ RL credit-C	10	77.0 ± 1.4 (+3.2)	45.4 ± 2.7 (+0.7)	87.6 ± 1.8 (+1.3)	51.8 ± 3.6 (+1.2)	+1.6 [−0.6, +3.8]
	20	78.1 ± 0.7 (+4.3)	46.4 ± 1.0 (+1.7)	87.5 ± 1.0 (+1.2)	54.0 ± 3.1 (+3.7)	**+2.7** [+0.6, +4.9]

Credit-C: crediting rules that survive. Credit-C also credits a call if it is the first to add a rule to memory and that rule appears in the union of verified rules from strictly later reflections in the episode. A rule can credit only its first author. Although verification may occur later, the extra weight is applied to the authoring call in its original window.

The Credit-B and surviving-rule conditions combine by union: a call meeting several conditions still receives raw weight 1 only once. Credit-C uses the same eligibility condition, weight rescaling in Equation 13, and unweighted scale σ as Credit-B. The resulting call advantages from all three variants are used in Equation 7.

Optimization and schedule. We run 30 training iterations with one optimization epoch per rollout batch. We use Adam with a constant learning rate of 10^{-6}, no warm-up, and a global gradient-norm limit of 1.0. The rollout engine receives the updated weights after each iteration. We log importance-ratio statistics as a numerical health check: both log-probabilities in the ratio are computed by the training implementation; under on-policy updates the ratio deviates from 1 due to trainer-side numerical noise. We apply no importance-sampling correction for the residual mismatch between the sampling engine and the trainer. Checkpoints are selected on the eight validation games. Validation uses 3 evaluation seeds, and the held-out test evaluation uses 5.

B.4 OOD Generalization

Transfer to external benchmarks. To test whether the gains extend beyond grid puzzles, we evaluate the base model and the checkpoints at iterations 10 and 20 on four external benchmarks of rule inference and hypothesis testing (Section 5), all with thinking on and a 32k-token cap. Table 7 reports each score with its paired difference to the base model and a paired-bootstrap 95% confidence interval. At iteration 20, the broadcast checkpoint improves the mean over the four benchmarks by 4.1 points (95% CI 2.1 to 6.2), with per-benchmark intervals excluding zero for CipherBank (+4.8) and WILT (+2.3), a lower bound at zero for PhysGym (+3.4), and an interval including zero for Failing-to-Falsify (+6.1). The credit-C checkpoint at iteration 20 improves the mean by 2.7 points (95% CI 0.6 to 4.9). At iteration 10, the mean improvements of both checkpoints are smaller and their intervals include zero. These results indicate that part of what is learned on WitnessGym transfers to rule inference outside the suite, with the clearest gains on cipher decryption and experiment design; the size of the effect is modest and the per-benchmark evidence is uneven.

B.5 Mis-behaviors in OSS Harness

Harness and evaluation set. The canonical evaluation harness presents the model with a human-readable board state. The model submits one action per turn, and each `CONFIRM` or `RESET` counts as a turn. Code execution and programmatic search are strictly prohibited. In the *OSS harness: Prime-Agent* configuration (Table 2), the agent instead receives raw 64×64 image frames and has access to a persistent Python REPL and an HTTP interface to the environment referee. These interfaces allow code to parse frames, submit actions, and use referee feedback to test candidate paths.

We rescore the Prime-Agent rollouts to distinguish level completion from the direct rule inference and action selection evaluated by the canonical harness.

The evaluation covers 17 private test games. It includes 34 rollouts each for Opus-5 and 4.8, and 85 rollouts each for Kimi-K3 and Qwen3.8-27B. Here *mis-behavior* denotes a departure from the benchmark's intended measurement construct, not necessarily meaning a violation of the coding agent's system instructions.

Model-attended scoring. The model-attended score requires model participation in each credited level. We exclude a level completion if it meets any of the following criteria:

- **Unattended Play (N):** No model forward pass occurs between the previous level completion and the current one. For example, a script solves the next level without the model observing its state or choosing its moves.
- **Programmatic Brute-Forcing (B1):** One code-cell execution produces three or more failed `CONFIRM` submissions. This threshold was set before rescoring. One or two failures in a cell do not trigger B1.
- **Background Execution (B2):** A background thread continues to submit actions after the code cell that started it has returned.

N concerns whether the model was called again before a level was completed. **B1** concerns repeated failed submissions within a single code-cell execution; failures from separate cells are not combined to reach its threshold. **B2** concerns actions submitted after control has returned from the initiating cell. For example, if a model starts a loop that clears several consecutive levels, the first completion can receive credit only if it satisfies the other criteria. Later completions without another model forward pass are excluded under N. A cell that repeatedly submits paths and resets after failures until one succeeds meets B1 once it produces three failed `CONFIRM` submissions. These cases differ from the model observing a failure and selecting the next attempt itself.

Verbal-only scoring. The verbal-only score applies the model-attended exclusions and adds one more criterion. We exclude a level under **S** if code generates its winning action sequence through search, enumeration, empirical rule-fitting, or reuse of a solver. Copying a path printed by code also triggers **S**. The exclusion applies even if the model wrote the solver or proposed the underlying rule: the program selected the moves used to complete the level.

We trace the provenance of each submitted action list. A level receives verbal-only credit if and only if the model explicitly typed the winning sequence, did not take it from prior code output or a programmatic solver, and the typed sequence matches the submitted moves. Code used solely to parse image frames is permitted under both scores, since the canonical harness already supplies parsed, human-readable boards.

Cascading Penalties (Rule B). For each score, we identify the first excluded level in a rollout. Under the preregistered strict penalty, Rule B, that level and all subsequent levels receive zero credit. A later level may therefore receive zero even if its own actions were model-attended. The rule prevents later completions from receiving credit when they may have benefited from rules or solvers acquired on an excluded level.

Environment probing and rollout termination. Prime-Agent models used their tools to parse image frames, retain solvers in memory, and run scripts across levels. We also observed models inspecting local directories, searching system files for game rules or source code, listing active processes and ports, querying undocumented referee endpoints, and attempting network requests or package installations. None of these probes successfully retrieved privileged game information. An attempted probe alone therefore does not exclude a level; exclusions follow the criteria above.

Rollouts also encountered failed context summaries, timeout errors, dead kernels, and token or session limits. The reported scores reflect the point at which each rollout terminated.

B.6 PROMPTS

System prompt (identical for every LLM call of a game, up to a per-call "Known Action Semantics" block summarizing the agent's own observations that the harness appends; the action placeholders are resolved to the game's action set, ACTION1–5 for all Witness games, and the plan cap to 16). Besides the coordinate convention in P4, the three trailing paragraphs are the only game-mechanic facts the agent is told: which button confirms, what RESET does, and the plan-length cap.

You are an interactive reasoning agent playing a puzzle video game. Each turn you receive a TEXT observation: an ASCII board with a legend and, on games that expose them, ground-truth lists of the board's markers and cell content, plus your recent actions with their effects and your own knowledge rules. Your goal: discover the rules and solve each level efficiently.

Core Priors

P1 OBJECTNESS: The environment has discrete bounded objects (the board elements named in the legend and, when present, the cell-content list). Objects persist between steps unless explicitly changed. They have: color, position, size, shape. Objects can appear, disappear, or change — track these transitions.

P2 AGENTNESS: You control something in the environment. Your actions (ACTION1-ACTION5) cause state changes. You must discover WHAT each action does by observing before/after. Goal: reach a winning state (the environment will signal level completion).

P3 NUMBERS: Objects can be counted. Quantities may be constraints (exactly N, at least N, at most N). Arithmetic relationships between quantities may matter.

P4 GEOMETRY: Spatial relationships matter — adjacency, containment, alignment, symmetry (horizontal/vertical/rotational). Distances can be Manhattan or Euclidean. Boundaries may block, wrap, or reflect. Coordinate system: (row, col), 0-indexed, (0,0) at TOP-LEFT — the same order as every coordinate in the observation. On a dot board the cursor moves on dot intersections; cell (r,c) is the square between dots (r,c), (r,c+1), (r+1,c) and (r+1,c+1).

Hypothesis-Test Protocol

Each turn:

1. OBSERVE: Read the current state carefully
2. HYPOTHESIZE: What rule explains the observations so far?
3. PREDICT: What will my chosen action cause?
4. ACT: Choose the most informative action
5. VERIFY: Was prediction correct? Update hypothesis confidence.

Exploration Strategy

- Simple first: try basic actions before complex strategies
- Systematic: test each available action from new states
- Contrastive: different actions in the same state reveal what each action does
- Errors are signals: negative feedback (visual changes, state resets) = direct evidence of constraint boundaries
- Hypothesis-driven: once you have a hypothesis, test it efficiently
- AVOID CYCLES: if you've been to a state before, try a DIFFERENT action
- Map actions early: in the first few steps, try each of ACTION1-4 to learn what they do

Reasoning Principles

- Occam's Razor: prefer the simplest hypothesis that explains all observations
- One rule at a time: test hypotheses individually, not in combination
- Distinguish mechanics from goals: HOW things work vs WHAT you need to achieve
- Failed attempts are the most informative: they reveal constraint boundaries
- Look for patterns across levels: rules often stay the same, only configurations change

HARD RULES — Mandatory Constraints

These rules are ABSOLUTE. Violating any of them wastes budget and degrades performance.

Rule Discovery

- R1: Output AT MOST 2 new rules per reflection. Quality over quantity.
- R2: DELETE any rule where counter-evidence $\geq$ supporting evidence. Do not keep disputed rules alive.
- R3: Rules below 0.2 confidence are dead — stop referencing them.
- R4: Maintain AT MOST 15 active rules. If at capacity, delete the weakest before adding.
- R5: NEVER create a rule from a single observation. Require at least 2 supporting observations.
- R6: Correlation is NOT causation. "X happened after Y" does not mean Y caused X — verify with a deliberate test.
- R7: If uncertain about a rule, cap its confidence at 0.5.

Planning

- P1: Plans must be AT MOST 16 actions. Longer plans accumulate prediction errors.
- P2: Each action in a plan MUST have a justification grounded in a known rule.
- P3: If 3 consecutive actions in a plan result in noop, ABANDON the plan immediately.

Global

- G1: When budget $> 95\%$ consumed, STOP exploring and crystallize knowledge immediately.
- G2: If 10 consecutive actions produce no state change, SWITCH strategy (different actions, different region).
- G3: When the same (state, action) produces different results, this is COGNITIVE DISSONANCE — trigger immediate reflection to resolve the contradiction.

ACTION5 is the CONFIRM/SUBMIT action: it does not move anything — press it to submit your COMPLETED solution and win the level. Plan it as the FINAL action, and only once your path satisfies every rule (confirming an incomplete/incorrect path fails the attempt).

ACTION0 is RESET: it restarts the CURRENT level from scratch. Your drawn path/progress on the board is erased, but everything you have LEARNED (rules, action semantics) is kept. IMPORTANT: all actions you have already taken on this level still count toward your step budget and efficiency score — resetting does not refund them. Use it deliberately, when your current path is unsalvageable and restarting is cheaper than undoing it move by move.

PLAN LENGTH: at most 16 actions per plan are executed; anything beyond the first 16 is dropped. Plan in chunks of at most 16 actions and continue from the resulting state.

User prompt of every LLM call (one call = one reflection: analyze, edit the rule memory, optionally emit a plan of up to 16 actions that the harness then executes). Slots in angle brackets are filled by the harness from the engine's ground truth and the agent's own memory; nothing in them reveals a hidden rule. The 'Harness notices" block appears only when the harness ignored or altered one of the agent's operations;' Recent Attempts" once a plan has finished on the current level (harness trial memory); the warning block only when a rule has conflicting evidence.

You are analyzing an unknown game environment to discover its rules and mechanics.
You must figure out everything from observation — no prior knowledge of this game exists.

Current Board State
⟨compact board text: grid type and size, 'Agent at row=r, col=c",' Level: k/M", the coordinate convention, a legend naming each symbol on the board, the ASCII board, Start/End coordinates, the mandatory dots and whether each is visited, on games that expose them the cell contents (e.g. square colours), 'Path so far", the per-action test status ('ACTION1: untested", 'ACTION4 (RIGHT): noop=0% [observed $\times 3$ this level]" or' ACTION5 (CONFIRM/SUBMIT): pressed $\times 1$ this level, 1 rejected"), and "Last change: agent moved from (r_1, c_1) to (r_2, c_2)" (or the outcome of a CONFIRM/RESET)⟩

Known Action Semantics
⟨per action, the direction, no-op rate and observation count the agent has itself established, plus the CONFIRM press count⟩

Recent Observations (last n steps)
⟨'No informative observations yet." or: the most recent observed movement per action, followed by the last actions (up to 8) and their effects, e.g.#3 ACTION5: submit REJECTED — path reset to start (2,0)"; a step the harness drew at random after a planless reflection is labelled as such⟩

Current Knowledge (n/15 rules)
⟨"No rules discovered yet." or the agent's numbered rules' [i] ⟨rule text⟩", a rule with confidence ≥ 0.8 carrying a ✓, a rule with conflicting evidence carrying ⚠DISPUTED, a rule kept from the previous level carrying [carried]⟩

[only when the harness ignored or altered one of the agent's operations]
Harness notices (since your last reflection)
⟨one line per event, e.g. "plan cap: your plan had 18 actions; only the first 16 ran — the last 2 were not executed"⟩

[once a plan has finished on this level]
Recent Attempts (this level)
You have made ⟨n⟩ attempts on this level so far.

- attempt #⟨i⟩: [⟨action ids⟩] → ⟨end reason, e.g. 'submit rejected at action 3/3" or' no submit — wandered without attempting; included 1 reset(s)"⟩ (last two attempts; a plan that ended without CONFIRM or RESET is listed as a "chunk" and not counted)
[if the last two attempts were the same actions from the same state] ⚠ Your last two attempts were IDENTICAL action sequences from the same starting state.
(Attempt history is provided automatically — do not store attempts as knowledge rules.)

Instructions
Analyze the recent observations in context of the board state and existing knowledge.
Look for:

- How actions affect the game state (movement patterns, special behaviors)
- Spatial patterns (board structure, special positions, boundaries)
- Win condition clues (what might lead to level completion)
- Effective strategies (sequences that make progress toward goals)

[only when a rule is disputed] ⚠ ATTENTION — The following rules have CONFLICTING evidence and may be wrong. Review carefully and consider deleting:
Rule [⟨i⟩]: "⟨rule text⟩" (support=⟨a⟩, contradictions=⟨b⟩)

Respond with:

1. `<meta>`Your concise analysis`</meta>`
2. Memory operations (one or more of these):
   `<add>`new rule or pattern you discovered`</add>`
   `<delete>`rule_number`</delete>` (to remove an outdated/wrong rule)
   `<keep/>` (if no changes to knowledge needed)
3. Optionally, if you have a specific strategy to try right now:
   Use `<plan>`...`</plan>` with action IDs as comma-separated numbers.

Reminders (full constraints in system prompt — R1-R7, P1-P3, G1-G3):

- Update knowledge via delete+add (don't duplicate existing rules); keep rules one sentence each.
- DISPUTED rules (⚠) have conflicting evidence — verify before relying.

B.7 Evaluation Details

Detailed evaluation settings including exact model version identifiers and reasoning budgets are given in Table 8

B.8 Pipeline

The full game creation pipeline is given in Algorithm 1.

Table 8: **Exact model identifiers and sampling settings behind Table 1.** Every model was run with the same canonical harness and protocol: 17 unseen games × 5 evaluation seeds × up to 20 levels; 300 actions per level (an unsolved level ends the rollout) and 1,000 LLM calls per rollout; `temperature` = 1.0 for all models (the client forces 1.0 whenever reasoning is enabled), no `top_p`, `top_k` or seed parameter, cached system prompt. *Reasoning*: "max" sends `reasoning:{effort:"max"}`; "16k budget" sends `reasoning:{max_tokens:16000}` (endpoints without effort tiers); "adaptive-max" is the Anthropic API's adaptive thinking with `output_config.effort=max`. *Out cap* is the per-call output-token limit.

Model	Requested identifier	Out cap / reasoning	Evaluated (2026)
Fable-5	`claude-fable-5`	64k / adaptive-max	Jul 31 – Aug 5
Opus-5	`anthropic/claude-opus-5`	32k / 16k budget	Jul 29 – Aug 5
GPT-6-Astra-Pro	`openai/gpt-6-astra-pro`	64k / max	Sep 9 – Sep 17
GPT-6-Astra	`openai/gpt-6-astra`	64k / max	Sep 4 – Sep 12
Kimi-K3	`moonshotai/kimi-k3`	32k / max	Jul 30 – Aug 6
GPT-5.6-Sol	`openai/gpt-5.6-sol`	32k / max	Aug 12 – Aug 22
Opus-4.8	`anthropic/claude-opus-4.8`	64k / max	Jul 30 – Aug 5
Muse-Spark-1.2	`meta/muse-spark-1.2`	32k / max	Aug 11 – Aug 19
Grok-4.6	`x-ai/grok-4.6`	32k / max	Aug 12 – Aug 17
DeepSeek V4 Pro	`deepseek/deepseek-v4-pro-0813`	32k / max	Aug 14 – Aug 19
Qwen-3.8-Max	`qwen/qwen3.8-2.4t-a95b`	32k / max	Aug 14 – Sep 4
Qwen3.8-27B	`qwen/qwen3.8-27b`	32k / max	Aug 17 – Aug 20
GLM-5.2-Max	`z-ai/glm-5.2`	32k / max	Jul 29 – Aug 5
GLM-5.3	`z-ai/glm-5.3`	32k / max	Aug 21 – Aug 25
Qwen3.5-35B-A3B	`qwen/qwen3.5-35b-a3b`	32k / 16k budget	Jul 29 – Aug 4
Glim-30	`meta/muse-glimmer-30b`	32k / max	Aug 11 – Aug 12
Dots-3-Note-Preview	`dots-studio/dots-3-note-preview:free`	32k / 16k budget	Aug 16 – Aug 17
Qwen3.5-9B	`qwen/qwen3.5-9b`	32k / 16k budget	Jul 29 – Aug 4

Algorithm 1 Game Creation Pipeline

Require: Natural-language rule concept
Ensure: Certified and packaged game bundle

Phase 1: Specification & Implementation
Generate natural-language spec with edge cases and pass neutrality tests.
$I_1, I_2 \leftarrow$ Independently generate two implementations from the spec.
Cross-examine I_1 and I_2 across a suite of test boards.
if $I_1 \neq I_2$ **then**
  Generate counter-examples and retry (up to 4 rounds) until zero disagreements.
end if
Merge I_1 and I_2 into a single certified module M.

Phase 2: Qualification Exams
Verify M for: necessity, visibility, non-triviality, and novelty against existing rules.

Phase 3: Level Generation & Filtering
$L_{\text{cand}} \leftarrow$ Generate a large pool of candidate levels.
$L_{\text{valid}} \leftarrow \emptyset$
for each level $l \in L_{\text{cand}}$ **do**
  Require: l passes structural validation and optimal solution depth ≥ 5
  Require: l necessitates rule mechanics (no heuristic shortcuts)
  Require: Win rate is 0 for all greedy baselines and ≤ 0.004 for random play
  if l satisfies all requirements **then**
    $L_{\text{valid}} \leftarrow L_{\text{valid}} \cup \{l\}$ ▷ De-duplicated via canonical hash
  end if
end for

Phase 4: Sequence Assembly & Teaching
Measure and profile kill-able wrong theories across L_{valid}.
Schedule levels S monotonically: Introduce $\rightarrow$ Accumulate $\rightarrow$ Reinforce.
Verify S against the shipped bank for schedule conformance and feature monotonicity.
Embed visual feedback and exploration constraints.

Phase 5: Audit & Packaging
Conduct human blind-judging and adversarial algorithmic testing.
Bundle M and S into the final package.
Execute canonical sequential replay to confirm deterministic, hash-checked victories.
return Game Bundle