SCHEMA: DISCOVERING UNKNOWN ENVIRONMENTS VIA AGENTIC PROGRAM INDUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning to complete tasks in unfamiliar environments with unknown rules remains a key challenge for LLM agents. Current LLM agents often record their discoveries in prose, which may not provide a compact, explicit account of how the environment works. Inspired by how scientists organize observations into testable, predictive theories, we introduce **Schema**, an agent harness that organizes learning and action through *interactive program induction*. The LLM agent decides what to investigate and how to act, expressing its evolving understanding of the environment as executable programs. The harness consists of a persistent program workspace and a small set of interfaces for checking these programs against the interaction history, planning within them, and executing plans under step-by-step verification. Schema raises ARC-AGI-3 RHAE from 58.7% to 99.2% with the same base model, solves 100% of the public DiG-bench games, and reaches the median performance of the top-50 human players on MazeBench. Extensive analysis shows the effectiveness of Schema in unknown mechanism discovery, and ablations confirm the contribution of each component.

1 INTRODUCTION

Provided with detailed financial data, an LLM agent can quickly produce a comprehensive summary. Given trip destinations and a budget, it can easily work out a thoughtful itinerary. Armed with established algorithms, it can even beat top human competitors in programming contests (OpenAI, 2025). Modern LLM agents, built on frontier models and equipped with advanced tools, now excel at tasks with familiar settings, known procedures, and well-defined objectives (Jimenez et al., 2024; Merrill et al., 2026; Xie et al., 2024; Yao et al., 2024; El-Kishky et al., 2025; Chan et al., 2025; Paglieri et al., 2025; Kwa et al., 2025). Yet acting in unknown environments where no explicit rules or instructions are given remains a challenge (Chen et al., 2026a;b; Phan et al., 2025). For example, on benchmarks that require LLM agents to learn a completely novel environment from scratch, even the strongest models still fall short of human performance (Warrier et al., 2025; Battleday et al., 2026a; Pappas & Pappas, 2026a).

This gap reflects a difference in how LLM agents and humans discover the mechanisms of an unknown environment. Humans do so through observation, experimentation, and reasoning (Gopnik & Wellman, 2012; Cook et al., 2011), and abstract what they find into explicit, reusable theories (Tenenbaum et al., 2011; Rule et al., 2020). In scientific discovery, for example, researchers derive predictions from theories, test them through experiments, and revise the theories in light of new evidence. Such theories transfer across situations (Lake et al., 2017), letting humans act efficiently and adapt quickly to new challenges (Tsividis et al., 2021; Dubey et al., 2018). LLM agents, by contrast, hold what they learn as prose in a context window or in editable memory. As context is compacted and memory rewritten (Packer et al., 2023), that prose degrades (Liu et al., 2024), and an agent that does not distill it into a persistent, reusable form is left disoriented in a complex, new environment (Laban et al., 2026; Backlund & Petersson, 2025).

Inspired by this observation, we present **Schema**, the first agentic harness that achieves **human-level** mechanism discovery in unknown environments. Schema expresses its understanding of environment mechanisms in the formal language of computers: programs. To organize how an agent constructs, tests, refines, and uses this understanding, we propose **interactive program induction**, a paradigm for agent design that Schema instantiates. Concretely, the LLM agent writes programs

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.

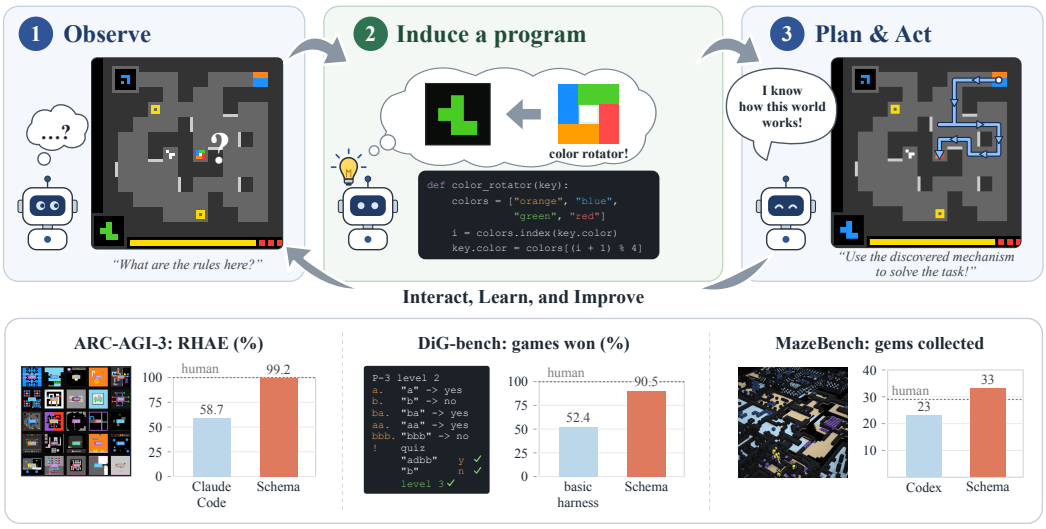


Figure 1: Schema observes the environment, encodes discovered mechanisms as executable programs, and uses them to guide planning and action, refining its understanding through further interaction (top). Schema achieves strong performance across all three benchmarks, outperforming baseline harnesses using the same base models (bottom); dashed lines indicate human reference performance. (a) ARC-AGI-3: RHAE over the 25 public games, Claude Fable 5 in Claude Code and in Schema; the human baseline is 100 by construction. (b) DiG-bench: public games won out of 21, GPT-6 Astra. (c) MazeBench: gems collected, GPT-6 Astra in Codex and in Schema, against the top-50 human median.

that encode environment representations and transitions, checks them against its interaction history, and uses them to guide experimentation and planning (Figure 1). This approach allows Schema to keep and continually update what it learns in a form that is persistent, executable, and verifiable.

We evaluate Schema's generality across environments and base models, efficiency in discovering hidden mechanisms, and ability to sustain progress over long horizons. On ARC-AGI-3 (ARC Prize Foundation, 2026a), Schema improves action efficiency across four model configurations, reaching 99.2% RHAE compared with 58.7% for the same base model in its coding harness. On DiG-bench (Battleday et al., 2026a), Schema discovers hidden rules with fewer failed attempts and solves all 21 public games. On MazeBench (Pappas & Pappas, 2026a), Schema accumulates and reuses knowledge across tens of thousands of interactions, sustaining exploration and reaching the median performance of the top-50 human players. Our contributions are as follows:

- **Paradigm.** We propose interactive program induction, a paradigm for designing LLM agents that learn and act in unknown environments by continually constructing, testing, and using executable programs.
- **Harness.** We build Schema, which instantiates this paradigm and reaches human-level performance on various novel environments.
- **Analysis.** Through ablations and behavioral analyses, we show how Schema's components support mechanism discovery and sustained exploration.

2 Preliminaries

Task Setting. Consider an agent acting in an unfamiliar environment whose available actions are given, but whose rules and completion conditions must be discovered through interaction. At step t, the agent receives an observation o_t of latent environment state s_t and chooses an action $a_t \in \mathcal{A}$. The environment evolves according to $s_{t+1} = T(s_t, a_t)$ and returns o_{t+1}, including task feedback. The unknown transition function T and completion predicate G, with $G(s) = 1$ indicating success, together specify the environment's *mechanisms*. The agent accumulates the ordered interaction history $\mathcal{H}_t = ((o_i, a_i, o_{i+1}))_{i<t}$. Its objective is to complete the task using as few interactions with the environment as possible (an objective of broad practical relevance, whether imposed by

game rules or motivated by the cost of real-world experimentation). Actions therefore serve both to advance the task and to provide evidence about the environment's mechanisms.

Program Induction. Program induction seeks an executable program P consistent with a set of input–output examples $\mathcal{D} = \{(x_i, y_i)\}$, so that $P(x_i) = y_i$ for each example. For example, from the pairs `abc` $\rightarrow$ `cba` and `cab` $\rightarrow$ `bac`, a learner may infer a program that reverses its input string. Standard counterexample-guided inductive synthesis organizes this search as a dialogue between a synthesizer and a verifier (Solar-Lezama et al., 2006; Solar-Lezama, 2008). The synthesizer proposes a candidate consistent with the current examples. The verifier checks it against a specification and returns a counterexample when it fails. Adding that counterexample to $\mathcal{D}$ constrains the next candidate while retaining the evidence already collected. For environment learning, action outcomes provide examples from which language models can induce programs describing the underlying dynamics (Tang et al., 2024; Dainese et al., 2024). Executing these programs makes their predictions available for both checking and simulation. The choice of which actions to take thus becomes part of the induction process.

3 Schema: Agentic Harness with Interactive Program Induction

We propose *interactive program induction* (IPI), a design paradigm that integrates program induction and action selection into a single LLM agent. Schema instantiates IPI through the four basic operations of *hypothesize*, *certify*, *plan*, and *act with verification* (Figure 2). Schema's harness consists of a persistent workspace and a set of interfaces for carrying out these operations (Appendix B). The LLM is free to organize calls to these interfaces, drawing on its broad prior knowledge to make context-dependent decisions that guide its interaction with the environment.

3.1 Hypothesize: Encoding Mechanisms as Programs

The Schema agent encodes its current understanding of the environment in a program P, which typically includes two components. (i) *State grounding* constructs a structured representation of observed objects, attributes, and relations, together with relevant information carried over from earlier interactions. (ii) *Transition rules* encode the effects of actions and object interactions on this representation. The harness invokes P through a common prediction interface

$$(\hat{o}_{t+1}, z_{t+1}) = P(z_t, o_t, a_t), \tag{1}$$

where z_t is the program's internal state and $\hat{o}_{t+1}$ is its prediction of the next observation. The agent uses file management tools to revise the program's state representation and transition rules, while the harness appends each observed transition to the interaction history.

3.2 Certify: Checking Consistency with History

A proposed program is checked by replaying the interaction history. For each recorded action, the harness evaluates Equation 1 using the recorded observation and the internal state reconstructed under P. Let $\hat{o}^P_{i+1}$ denote this prediction, and write $\hat{o}^P_{i+1} \simeq o_{i+1}$ when its observable consequences agree with the environment's output, including completion or failure feedback. The mismatch set and history-consistency criterion are

$$\begin{aligned} \mathcal{M}(P, \mathcal{H}_t) &= \{\, i < t : \hat{o}^P_{i+1} \not\simeq o_{i+1} \,\}, \\ \mathrm{Cert}(P, \mathcal{H}_t) &\iff \mathcal{M}(P, \mathcal{H}_t) = \emptyset. \end{aligned} \tag{2}$$

In Schema, the harness provides backtesting tools that report mismatches together with predicted and observed outcomes, giving the agent concrete counterexamples from which to revise its hypothesis. Rechecking the full history tests each revision against previously explained behavior, allowing discoveries to accumulate in a coherent program. In practice, the LLM iteratively uses file management and backtesting tools to revise the program and check its consistency with the full interaction history.

3.3 Plan: Searching within the Program

The same prediction interface makes P a simulator: feeding predicted observations and internal states back into Equation 1 produces a rollout without interacting with the environment. The agent specifies a target predicate g over these simulated states and observations. The target may express

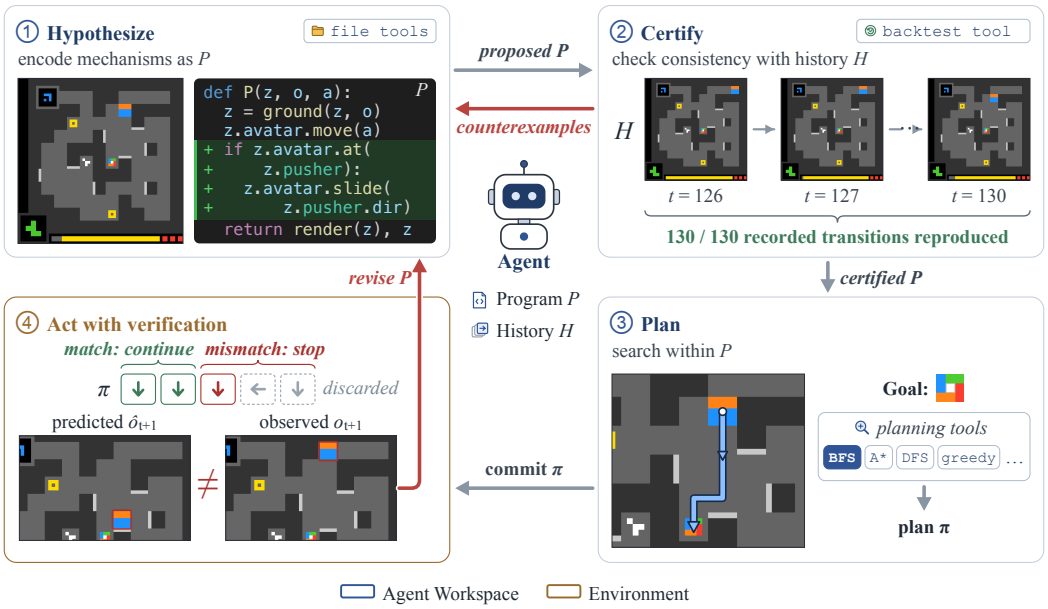


Figure 2: **Overview of Schema.** The LLM agent expresses its understanding as an executable program P in a persistent workspace. The harness checks P against recorded interactions and searches within it using goals and procedures chosen by the agent, without consuming environment actions. Committed plans execute one action at a time under prediction checks. A mismatch interrupts execution and returns new evidence for revising P. The ARC-AGI-3 LS20 level 3 example illustrates an unmodeled color-rotator effect with a simplified prediction: the key is predicted to remain orange but turns blue.

task completion, an intermediate waypoint, or a situation in which an unresolved mechanism can be tested. Planning therefore serves both task progress and experiment design. Given a target, the agent chooses a search procedure σ and computational budget b and invokes

$$\pi \leftarrow \textsc{Search}_\sigma(P, z_t, o_t, g; b), \qquad \pi = (a_t, \ldots, a_{t+k-1}). \tag{3}$$

The Schema harness runs the selected procedure inside P and returns a candidate plan or reports that none was found within the budget. The agent can call built-in planning tools (e.g., breadth-first, depth-first, A*, or greedy search) or write and load any search procedure of its own. It also sets and can adjust search depth, node limits, and runtime in its tool calls. The returned plan and search feedback help the agent decide whether to commit to the actions, refine the search, or investigate further.

3.4 Act with Verification: Testing Predictions in the Environment

The agent commits an action sequence of its chosen length, obtained through search or designed directly as an experiment. Before each action a_t, the harness obtains the program's prediction using Equation 1; it then executes the action and appends the observed transition to $\mathcal{H}_t$. If prediction and observation agree, execution proceeds without another LLM call. Inspired by model predictive control (Li & Shi, 2014), the harness discards the remaining actions at the first prediction mismatch and returns the unexpected transition to the agent. Each executed action extends the evidence available for induction. The agent is free to use the returned observations to decide whether to revise the program, adjust its plans, or investigate further.

4 Experiments

Our experiments assess Schema's *generality across games and base models*, *efficiency in discovering hidden mechanisms*, and *ability to sustain progress over long horizons*. ARC-AGI-3 establishes performance across base models and uses component ablations to explain how discoveries translate into efficient action (Section 4.1). DiG-bench examines the cost of identifying hidden rules, measured by games won and lives lost at different reasoning efforts (Section 4.2). MazeBench examines

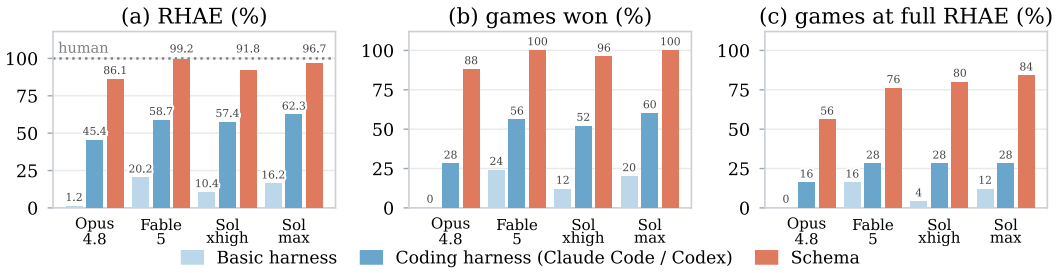


Figure 3: **Schema improves completion and action efficiency across model configurations.** Results on the 25 public ARC-AGI-3 games: (a) RHAE, (b) percentage of games won, and (c) percentage of games achieving 100% RHAE. Each model configuration is evaluated with the basic harness, coding harness, and Schema.

whether knowledge remains compact and useful as interaction history grows, linking program reuse to continued exploration (Section 4.3). Within each benchmark, we compare Schema with baseline harnesses using the same base model. Further benchmark background and scoring details are given in Appendix C, and gameplay videos are in the supplementary material.

4.1 ARC-AGI-3: Human-level Action Efficiency

ARC-AGI-3 requires agents to discover unfamiliar game mechanics through interaction and apply them to increasingly complex levels. We assess Schema's performance relative to human players and baseline harnesses, then use component ablations to analyze the source of its gains.

Task Setup. ARC-AGI-3 (ARC Prize Foundation, March 2026) contains 25 public games, each with six to ten levels presented as 64×64 color grids. The agent is given up to seven available actions and must discover their effects and the game's objective through interaction. The primary metric, relative human action efficiency (RHAE), combines level completion with action efficiency relative to first-time human players (Appendix C.1). We also report the number of games won and games reaching the maximum score. The benchmark remains challenging: in its official basic harness, models whose training-data cutoffs precede its public release still only achieve at most 30% RHAE. We evaluate Claude Opus 4.8 (Anthropic, 2026e), Claude Fable 5 (Anthropic, 2026b), and GPT-5.6 Sol (OpenAI, 2026d), with Sol tested at both extra-high and maximum reasoning effort. We compare Schema with two types of baseline harnesses using the same base model: (i) the official ARC-AGI-3 basic harness, which provides no auxiliary tools, and (ii) the model's coding harness (Claude Code (Anthropic, 2026a) for Claude models and Codex (OpenAI, 2026a) for GPT models), which allows agents to maintain persistent notes and execute code and shell commands.

Table 1: Ablation results.

	RHAE (%)
Schema (full)	**72.9**
prose model	58.8
w/o certification	62.3
w/o planning	59.2
w/o verification	52.4

Main Results. Across the four configurations, Schema raises RHAE from 1.2–20.2% in the basic harness to 86.1–99.2% and increases games won from 0–6 to 22–25 (Figure 3). Relative to the coding harnesses, Schema improves RHAE by 34.4–40.7 percentage points (Figure 3a). It reaches 99.2% with Fable 5, 96.7% and 91.8% with Sol at maximum and extra-high effort, and 86.1% with Opus 4.8, compared with 58.7%, 62.3%, 57.4%, and 45.4% in their coding harnesses. With Fable 5 and Sol at maximum effort, Schema clears all 25 games, compared with 14 and 15 for the respective coding harnesses; 19 and 21 games reach the maximum score, compared with seven for each baseline (Figure 3b,c). With Fable 5, Schema completes every one of the 25 games in fewer actions than the human baseline, even using just 57% of the human action total across all 183 levels. These results suggest that frontier models do not lack the ability to understand unfamiliar environments. However, they need the right harness to put their broad knowledge to use.

Ablation Study & Analysis. To further understand how the harness helps models put their knowledge to use, we replace the executable program with prose and separately remove the interfaces for certification, planning, and verified execution from Schema. We evaluate all variants with Claude Opus 4.8 on the ten hardest games, ranked by the number of actions in their human baselines, cover-

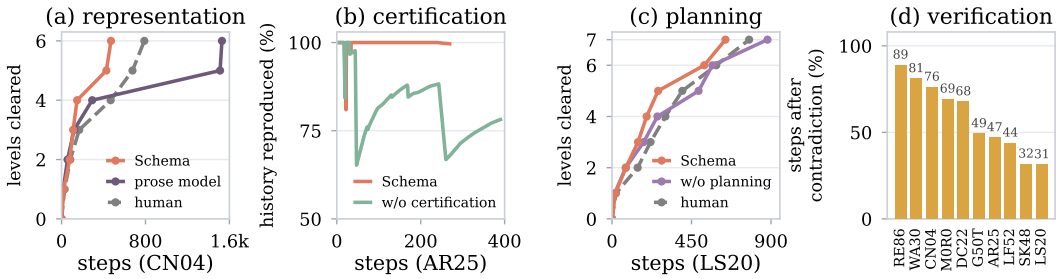


Figure 4: **Trace dynamics.** (a) Levels cleared versus cumulative steps on CN04 with program and prose representations, alongside humans. (b) Offline history agreement on AR25 with and without certification, using Schema's Opus 4.8 main-evaluation run. (c) Levels cleared versus cumulative steps on LS20 with and without planning, alongside humans. (d) Steps taken after a prediction mismatch within the same plan without verification.

ing 75 levels with a budget of 2,000 actions per game. As shown in Table 1, removing any of these components substantially reduces performance, with RHAE falling from 72.9% to 52.4–62.3%.

We further examine the agents' reasoning and interaction traces to understand how these ablations affect their discovery and use of the environment's rules, with detailed cases in Appendix G. (i) **Program induction makes beliefs explicit and reusable.** In CN04 level 5, pieces must be moved and rotated to pair their connectors without overlapping, while one piece grows and changes the geometry. The prose agent repeatedly revisits the connection rules; Schema enumerates arrangements under explicit geometric constraints, completing the level in 279 rather than 1,219 actions (Figure 4a). (ii) **Certification helps the agent validate hypothesis correctness.** In AR25, the agent moves pieces and mirror axes so that reflected shapes match a target. Without certification, a revision to identify the active piece improves predictions on the current level but drops history agreement from 97.7% to 65.2%; Schema's final model reproduces 99.6% of its history (Figure 4b). (iii) **Planning ahead improves action efficiency.** Even with a program that fits observed transitions, choosing an efficient action sequence requires reasoning about future states. In LS20 level 5, the agent must transform a carried symbol to match its target, timing contact with a moving rotation marker within an energy budget. Without planning, the model passes all 252 transition checks at entry to the level, but the agent works out routes and transformations manually, taking 233 actions compared with 65 for Schema (Figure 4c). (iv) **Runtime verification further boosts adaptation to new scenarios.** In LS20 level 2, the agent must rotate a carried symbol and reach the goal using single-use refueling stations. Without verification, it assumes these stations are reusable and continues for 42 actions after the first mismatch, returning to a spent station and running out of energy. Across the ten games in Figure 4d, such continuations account for 58% of actions.

4.2 DiG-bench: Efficient Mechanism Discovery

We next turn to a setting where discovering hidden mechanisms is central to success. On DiG-bench, we examine whether Schema can choose informative interactions and efficiently infer the underlying rules.

Task Setup. DiG-bench (Battleday et al., August 2026) evaluates interactive mechanism discovery through text-based games across seven difficulty tiers. Game states are presented as short strings of letters, numbers, and symbols, but both the transition rules and the win conditions must be discovered through interaction. For example, in P-3, the agent queries whether chosen strings satisfy a hidden rule, then classifies new strings, with each incorrect quiz answer costing a life. The agent must choose informative actions and use the resulting feedback to identify hidden rules while completing challenges with very limited lives and per-level step budgets. Some games also offer a creative mode for experimentation, allowing the agent to construct situations that test its hypotheses. These games remain challenging for frontier models. The benchmark authors report that the strongest model achieves a 71.4% win rate overall and only 40% on the two hardest tiers, while every game was solved by at least one human on their first attempt. Existing agentic harnesses have also shown limited gains over the basic harness on these discovery tasks (Battleday et al., 2026a).

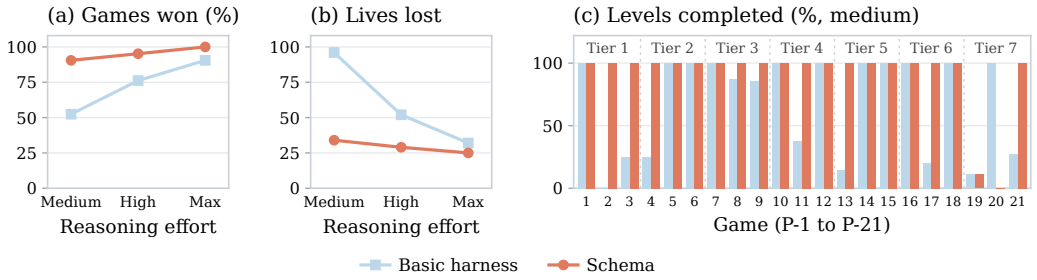


Figure 5: **Schema solves more games with fewer failed attempts.** GPT-6 Astra on the 21 public DiG-bench games: (a) game win rate and (b) total lives lost at each reasoning effort; (c) percentage of levels completed in each game at medium effort, grouped by difficulty tier.

We evaluate Schema on the 21 publicly available games, spanning all seven difficulty tiers, using GPT-6 Astra (OpenAI, 2026e) at medium, high, and maximum reasoning effort.

Results. Compared with the basic harness, Schema improves game completion and reduces failed attempts at all three reasoning efforts (Figure 5). It clears 19, 20, and 21 games at medium, high, and maximum effort, respectively, compared with 11, 16, and 19 for the basic harness. On the two hardest tiers (6–7), Schema raises the win rate at maximum effort from 66.7% to 100%. At medium effort, Schema already matches the basic harness's maximum-effort win count; compared with the basic harness at medium effort, it also reduces lives lost from 96 to 34. The same trends hold across repeated runs (Appendix E). Pooling costs and completed levels across all three reasoning efforts, Schema's estimated token cost per cleared level is $3.40, approximately 15% lower than the basic harness's $4.00 (Figure 6a). For levels cleared by both harnesses, the cost curve pooled by level index shows higher spending by Schema at early levels but lower mean costs at every index from 9 to 14 (Figure 6b). This pattern is consistent with reusing mechanisms discovered early in the game to solve later levels. Together, these results show that Schema supports efficient mechanism discovery with fewer failed attempts, a lower aggregate token cost per cleared level, and less reasoning effort to reach comparable game completion.

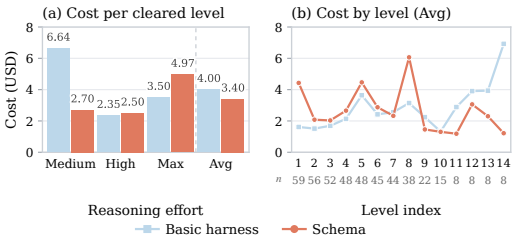


Figure 6: **Token cost on DiG-bench.** Average cost (a) per cleared level and (b) by level index, including input and output tokens at OpenAI's official API rates (OpenAI, 2026f).

Analysis. We examine the interaction traces and find that Schema is more systematic than the basic harness in testing its hypotheses. When feedback contradicts its current understanding, Schema revises the program and uses further queries to distinguish competing explanations, checking that the new rule also accounts for earlier observations. This helps it resolve uncertainty before risking lives on an untested answer. For example, in P-3 level 3, Schema tests strings with different letter orders and counts to discover the acceptance rule that the basic harness misses. See Appendix H for the queries, program revisions, and quiz outcomes in this level.

4.3 MazeBench: Knowledge Accumulation over Long Horizons

Finally, we examine whether Schema can build up and draw on its understanding of a large environment over long horizons. We observe sustained knowledge acquisition and agentic exploration.

Task Setup. MazeBench (Pappas & Pappas, July 2026) is a three-dimensional exploration and puzzle-solving environment with 256 interconnected rooms and 100 gems (Figure 7). Its rooms feature mechanisms such as lifts, bridges, springs, and carpets, alongside scenery ranging from forests and icy mountains to palaces and city walls; the rules governing these mechanisms must be discovered through interaction. We evaluate Schema with GPT-6 Astra, measuring exploration progress by gems collected and rooms visited, and compare its trajectory with that of the same

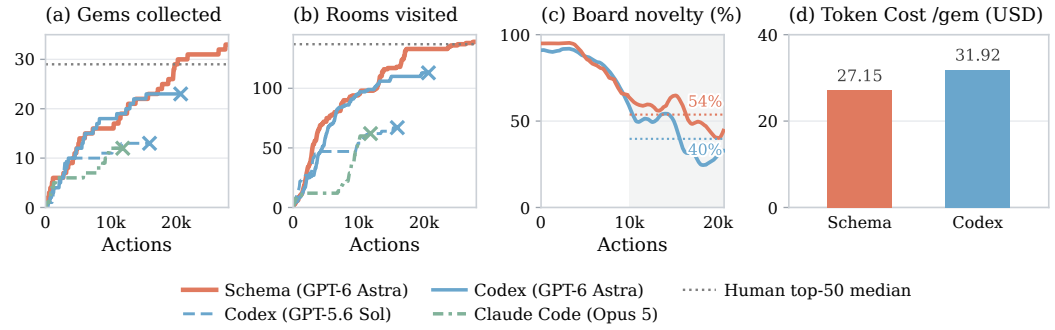


Figure 8: **Schema sustains progress over long horizons.** (a, b) Gems collected and rooms visited over environment actions; crosses mark the last recorded action of each baseline trajectory, and dotted lines show the top-50 human median. (c) Board-state novelty, with horizontal lines indicating each curve's mean over the shaded interval after 10,000 actions. (d) Token cost per gem at 23 collected gems.

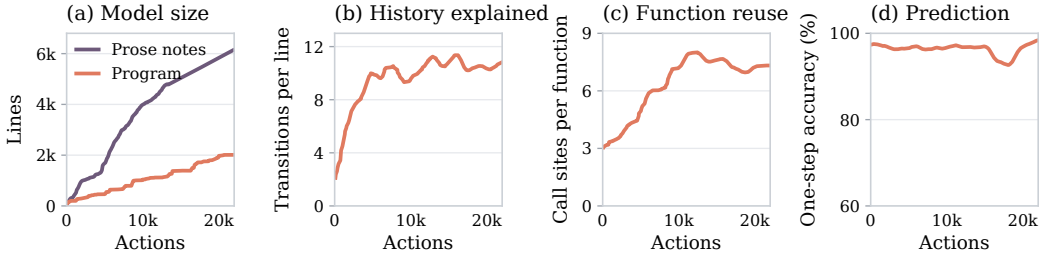


Figure 9: **Knowledge accumulates in a compact, reusable program.** (a) Non-comment program lines in Schema and note lines in the prose-model variant. (b) Recorded transitions explained per program line. (c) Call sites per function. (d) Forward prediction accuracy over the trailing 1,000 actions.

model in Codex, using human performance as a reference. At submission, the top-50 human players on the official leaderboard have an average recorded playtime of 37.2 hours and an average action count of 65,876; their median gem collection and room coverage are 29.0% and 53.5%, respectively, reflecting the demands of sustained exploration in this environment.

Results. Over a 36-hour run, Schema collects 33 gems and visits 139 rooms in 27,819 actions, exceeding the top-50 human median on both metrics (Figure 8a,b). At the same action count of 20,673, Schema has collected 30 gems and visited 133 rooms, compared with 23 gems and 113 rooms for the Codex baseline, at a similar token cost (Figure 8d). Both agents have visited 95 rooms by action 10,000; over the next 10,000 actions, Schema adds 38 rooms and 13 gems, while Codex adds 18 rooms and five gems. In the plotted interval after action 10,000, Schema also maintains higher mean board-state novelty, at 54% compared with 40% for Codex (Figure 8c). These results show that Schema sustains broader exploration and continues to make progress over long interaction horizons. Appendix F reports every gem's location and collection step, together with room coverage maps for all four runs in Figure 8.

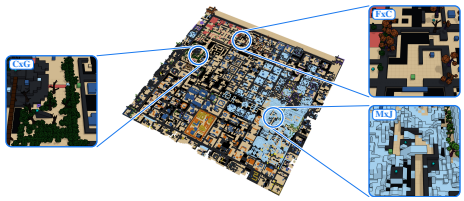


Figure 7: **MazeBench.** The full world of 256 rooms, with close-ups of rooms CxG, FxC, and MxJ.

Analysis. We examine how the agent (i) *accumulates knowledge in its program* and (ii) *reuses that knowledge in new situations during prolonged interaction* with an unfamiliar environment. Schema incorporates new experience by revising and extending the program's shared rules. By approximately 20,000 actions, the program contains 2,012 non-comment lines, compared with 6,133 lines

of notes in the prose-model variant (Figure 9a). The number of recorded transitions explained per line and the average number of call sites per function both increase, while forward prediction accuracy remains high (Figure 9b–d). These trends show that growing experience is captured in compact, reusable rules that retain their predictive value. These rules also support planning in new configurations after repeated context compaction. For example, on ice the player keeps sliding until blocked or reaching ordinary ground, so a route to the gem must account for where each slide stops. Schema learns this mechanism in room IxG at around action 2,600. Roughly 25,000 actions later, after at least 20 context compactions, it reuses the rule in a new ice maze in room NxE. After incorporating the new room's layout into the program, Schema uses A* search to find a 38-action route to the gem. It executes the entire route, with every observed transition matching the program's prediction (Appendix I).

5 Related Work

Agentic Harness. Agent harnesses shape how language models behave and what they can accomplish. ReAct (Yao et al., 2023), CodeAct (Wang et al., 2024c), and SWE-agent (Yang et al., 2024) demonstrate the importance of interaction loops and tool interfaces. Reflexion (Shinn et al., 2023) and ACE (Zhang et al., 2025b) enable agents to learn from experience by turning feedback into reusable reflections and strategies. Recent work further explores self-improving agents and harnesses by revising the agent's own code (Zhang et al., 2025a) and updating persistent prompts, memories, and skills (Karten et al., August 2026). Schema organizes learning and action around one persistent, evolving executable theory of the environment, continually certified against the full interaction history and directly used for experimentation and planning.

Code World Models. Programs provide interpretable, executable models of environment dynamics, an approach explored in theory-based reinforcement learning before the rise of LLM agents (Tsividis et al., 2021). WorldCoder (Tang et al., 2024) and CWM (Dainese et al., 2024) use LLMs and execution feedback to synthesize and refine Python world models. Subsequent work develops hierarchical planning in TheoryCoder (Ahmed et al., 2025) and compositional probabilistic representations in PoE-World (Piriyakulkij et al., 2025). These methods use LLMs as operators for generating and revising code within predefined learning procedures, such as WorldCoder's REx search, and typically learn dynamics over supplied state representations. Schema builds on these insights but makes the entire learning process agent-directed, transferring control over model revision, experimentation, and planning to the agent. The agent determines how to learn in environments where state grounding, transition rules, and task goals are all initially unknown.

Concurrent Work. Recent work has explored related approaches on ARC-AGI-3, including EWM (Rodionov, July 2026), OPINE-World (Courtis et al., July 2026), NOOA (Furgale et al., July 2026), Tycho (Lehmann et al., July 2026), and Twin (Skoutnev et al., August 2026). These systems combine executable modeling, interaction feedback, and planning, with some also citing earlier releases of Schema. These concurrent efforts highlight the promise of interactive program induction, which our study explores as a general approach to agent design across diverse unfamiliar environments.

6 Conclusion

We introduced interactive program induction, an approach in which an LLM agent learns about an unfamiliar environment by constructing, testing, and using executable models. Schema implements this approach through a persistent program workspace and tools for checking predictions against interaction history, planning, and verifying actions, while the agent directs experimentation and program revision. Across ARC-AGI-3, DiG-bench, and MazeBench, Schema improves task performance with frozen language models, supporting efficient mechanism discovery and sustained exploration over long interaction horizons. Ablations support the contribution of each component, while behavioral analyses illustrate how learned rules can persist across context compactions and be reused in new situations. Together, these findings suggest that persistent executable models provide a way for agents to accumulate knowledge over interaction, turning individual discoveries into explicit hypotheses that can be tested, refined, reused, and ultimately acted upon.

AI USE STATEMENT

In this work, we used generative AI tools to assist with figure-generation and to check bibliographic entries. We have not used generative AI tools to assist paper writing, to generate research ideas, to design experiments, or to produce or interpret results. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including figures or artifacts produced with the aid of generative AI.

REFERENCES

Zergham Ahmed, Joshua B. Tenenbaum, Christopher J. Bates, and Samuel J. Gershman. Synthesizing world models for bilevel planning. *arXiv preprint arXiv:2503.20124*, March 2025.

Zergham Ahmed, Kazuki Irie, Joshua B. Tenenbaum, Christopher J. Bates, and Samuel J. Gershman. Learning abstractions for hierarchical planning in program-synthesis agents. *arXiv preprint arXiv:2602.00929*, 2026. URL https://arxiv.org/abs/2602.00929.

Anthropic. Claude Code: Overview. https://code.claude.com/docs/en/overview, 2026a. Accessed September 25, 2026.

Anthropic. Claude Fable 5 model documentation. https://platform.claude.com/docs/en/models/fable-5/overview, 2026b. Released June 9, 2026.

Anthropic. Claude Fable 5.1 model documentation. https://platform.claude.com/docs/en/models/fable-5-1/overview, 2026c. Released September 1, 2026.

Anthropic. Claude Opus 4.7 model documentation. https://platform.claude.com/docs/en/models/opus-4-7/overview, 2026d. Released April 16, 2026.

Anthropic. Claude Opus 4.8 model documentation. https://platform.claude.com/docs/en/models/opus-4-8/overview, 2026e. Released May 28, 2026.

Anthropic. Claude Opus 5 model documentation. https://platform.claude.com/docs/en/models/opus-5/overview, 2026f. Released July 24, 2026.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, March 2026a.

ARC Prize Foundation. OpenAI's GPT-6 Astra on ARC-AGI-3. https://arcprize.org/blog/astra, 2026b. Accessed September 2026.

ARC Prize Foundation. Fable-class models score approximately 20% on the ARC-AGI-3 Public Demo environments. https://x.com/arcprize/status/2080716563448295720, 2026c. Post on X, July 24, 2026.

ARC Prize Foundation. Analyzing GPT-5.5 & Opus 4.7 with ARC-AGI-3. https://arcprize.org/blog/arc-agi-3-gpt-5-5-opus-4-7-analysis, 2026d. Blog post, May 1, 2026.

ARC Prize Foundation. ARC-AGI leaderboard. https://arcprize.org/leaderboard, 2026e. Accessed September 2026.

ARC Prize Foundation. Claude Opus 5: ARC-AGI results. https://arcprize.org/results/anthropic-claude-opus-5, 2026f. Accessed September 26, 2026.

ARC Prize Foundation. GPT-5.6 Sol: ARC-AGI results. https://arcprize.org/results/openai-gpt-5-6-sol, 2026g. Accessed September 26, 2026.

Axel Backlund and Lukas Petersson. Vending-bench: A benchmark for long-term coherence of autonomous agents. *arXiv preprint arXiv:2502.15840*, 2025.

Ruairidh M. Battleday, Kai Sandbrink, Jimi Cullen-Drohan, Zihan Yan, Timothy Muller, Clare Maguire, Ales Kubicek, Fraser Greenlee-Scott, Sukrit Sumant, Tri Dao, Jürgen Schmidhuber, Michal Valko, Joshua Tenenbaum, Thomas L. Griffiths, Zeb Kurth-Nelson, and James C. R. Whittington. DiG-bench: Discovery in games. *arXiv preprint arXiv:2608.12593*, August 2026a.

Ruairidh M. Battleday et al. DiG-bench: Discovering unknown rules in text-based games. https://digbench.ai, 2026b. Accessed September 26, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations (ICLR)*, 2025.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for LLM agents via environment interaction. In *International Conference on Learning Representations (ICLR)*, 2026a.

Shiqi Chen, Tongyao Zhu, Zian Wang, Jinghan Zhang, Kangrui Wang, Ruochen Zhou, Siyang Gao, Teng Xiao, Yee Whye Teh, Junxian He, and Manling Li. Why do LLM agents fail in exploring new environments? a world-modeling perspective. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026b.

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

David Courtis, Wenhao Li, and Scott Sanner. OPINE-World: Programmatic world modeling with ontology-error-prioritized interactive exploration for ARC-AGI-3. *arXiv preprint arXiv:2607.01531*, July 2026.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by Monte Carlo tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1349–1357, 2018.

Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*, 2025.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850, 2021. doi: 10.1145/3453483.3454080. URL https://doi.org/10.1145/3453483.3454080.

Alexis Fox, Junlin Wang, Paul Rosu, and Bhuwan Dhingra. PRO-LONG: Programmatic memory enables long-horizon reasoning. *arXiv preprint arXiv:2607.20064*, July 2026.

Paul Furgale, Severin Klingler, James Nolan, Matt Staats, Gaia Di Lorenzo, Elisa Martinez Abad, Christian Schüller, Razvan Dinu, Alessio Devoto, Pascal Berard, Gal Kaplun, Elad Sarafian, Riccardo Roveri, Leon Derczynski, and Ricardo Silveira Cabral. NVIDIA-labs OO agents: Native Python object-oriented agents. *arXiv preprint arXiv:2607.20709*, July 2026.

Kanishk Gandhi, Michael Y. Li, Lyle Goodyear, Agam Bhatia, Louise Li, Aditi Bhaskar, Mohammed Zaman, and Noah D. Goodman. BoxingGym: Benchmarking progress in automated experimental design and model discovery. *arXiv preprint arXiv:2501.01540*, 2025. URL https://arxiv.org/abs/2501.01540.

Google DeepMind. Gemini 3.1 Pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026. Published February 19, 2026.

Alison Gopnik and Henry M. Wellman. Reconstructing constructivism: Causal models, Bayesian learning mechanisms, and the theory theory. *Psychological Bulletin*, 138(6):1085–1108, 2012.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pp. 317–330. ACM, 2011. doi: 10.1145/1926385.1926423.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2.

Qiushi Han, Keya Hu, Linlu Qiu, Cathy Wu, and Kaiming He. VISTA: A visual harness for reasoning in an interactive world, August 2026. URL https://vista-research.github.io/. Blog post.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

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*, August 2026.

Zaid Khan, Archiki Prasad, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. One life to learn: Inferring symbolic world models for stochastic environments from unguided exploration. *arXiv preprint arXiv:2510.12088*, October 2025.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations (ICLR)*, 2026.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

Jens Lehmann, Andrei Aioanei, and Sahar Vahdati. Tycho: Active abstraction with programmatic world models for ARC-AGI-3. *arXiv preprint arXiv:2607.28287*, July 2026.

Huiping Li and Yang Shi. Event-triggered robust model predictive control of continuous-time nonlinear systems. *Automatica*, 50(5):1507–1513, 2014. doi: 10.1016/j.automatica.2014.03.015.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Moonshot AI. Kimi K3 model card. https://huggingface.co/moonshotai/Kimi-K3, 2026. Released July 2026.

OpenAI. OpenAI reasoning models solve all 12 problems at the 2025 ICPC world finals. https://x.com/OpenAI/status/1968368133024231902, 2025. Accessed September 2026.

OpenAI. Codex CLI. https://learn.chatgpt.com/docs/codex/cli, 2026a. Accessed September 25, 2026.

OpenAI. GPT-5.4 model documentation. https://developers.openai.com/api/docs/models/gpt-5.4, 2026b. Accessed September 2026.

OpenAI. GPT-5.5 model documentation. https://developers.openai.com/api/docs/models/gpt-5.5, 2026c. Accessed September 2026.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026d. Released July 9, 2026.

OpenAI. GPT-6 Astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, 2026e. Accessed September 2026.

OpenAI. GPT-6 Astra model. https://developers.openai.com/api/docs/models/gpt-6-astra, 2026f. API pricing. Accessed September 25, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations (ICLR)*, 2025.

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

Jonathan Pappas and David Pappas. Introducing MazeBench. https://mazebench.com/blog?post=introducing-mazebench, July 2026b. Published July 28, 2026.

Hanna M. Pasula, Luke S. Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007. doi: 10.1613/jair.2113.

Long Phan, Mantas Mazeika, Andy Zou, and Dan Hendrycks. TextQuests: How good are LLMs at text-based video games? *arXiv preprint arXiv:2507.23701*, 2025.

Wasu Top Piriyakulkij, Cassidy Langenfeld, Tuan Anh Le, and Kevin Ellis. Doing experiments and revising rules with natural language and probabilistic reasoning. *arXiv preprint arXiv:2402.06025*, 2024. URL https://arxiv.org/abs/2402.06025.

Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. PoE-World: Compositional world modeling with products of programmatic experts. *arXiv preprint arXiv:2505.10819*, May 2025.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.08559.

Qwen Team. Qwen3.6-27B model card. https://huggingface.co/Qwen/Qwen3.6-27B, 2026. Released April 2026.

Sergey Rodionov. Executable world models for ARC-AGI-3 in the era of coding agents. In *Artificial General Intelligence: 19th International Conference, AGI 2026, San Francisco, CA, USA, July 27–30, 2026, Proceedings, Part II*, Lecture Notes in Computer Science, pp. 198–210. Springer, 2026. doi: 10.1007/978-3-032-33195-3_15. arXiv:2605.05138, May 2026.

Joshua S. Rule, Joshua B. Tenenbaum, and Steven T. Piantadosi. The child as hacker. *Trends in Cognitive Sciences*, 24(11):900–915, 2020.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4.

SeungWon Seo, DongHeun Han, SeongRae Noh, and HyeongYeop Kang. Baba in Wonderland: Online self-supervised dynamics discovery for executable world models. *arXiv preprint arXiv:2605.16725*, 2026. URL https://arxiv.org/abs/2605.16725.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Alexy Skoutnev, Kirill Acharya, Gaston Longhitano, Madeleine Udell, Kevin Ellis, and Iddo Drori. Twin: Playing an unknown game with a test-time digital twin. *arXiv preprint arXiv:2608.14490*, August 2026.

Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California, Berkeley, 2008. Technical Report UCB/EECS-2008-177.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 404–415. ACM, 2006. doi: 10.1145/1168857.1168907.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML)*, pp. 216–224, 1990. doi: 10.1016/B978-1-55860-141-3.50030-4.

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Joshua B. Tenenbaum, Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

Pedro A. Tsividis, Joao Loula, Jake Burga, Nathan Foss, Andres Campero, Thomas Pouncy, Samuel J. Gershman, and Joshua B. Tenenbaum. Human-level reinforcement learning through theory-based modeling, exploration, and planning. *arXiv preprint arXiv:2107.12544*, 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. Hypothesis search: Inductive reasoning with language models. In *International Conference on Learning Representations*, 2024b. URL https://arxiv.org/abs/2309.05660.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning*, 2024c.

Archana Warrier, Dat Nguyen, Michelangelo Naim, Moksh Jain, Yichao Liang, Karen Schroeder, Cambridge Yang, Joshua B. Tenenbaum, Sebastian Vollmer, Kevin Ellis, and Zenna Tavares. Benchmarking world-model learning with environment-level queries. *arXiv preprint arXiv:2510.19788*, 2025.

xAI. Grok models and pricing. https://docs.x.ai/developers/models, 2026. Accessed September 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Z.ai. GLM-5.2: Built for long-horizon tasks. https://huggingface.co/blog/zai-org/glm-52-blog, 2026. Released June 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, May 2025a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, October 2025b.

A Extended Related Work

A.1 Agent Harnesses and Knowledge Representation

An agent's harness shapes both how it interacts with the world and what it carries forward from those interactions. ReAct (Yao et al., 2023) interleaves reasoning and actions, CodeAct (Wang et al., 2024c) uses executable code as an action interface, and SWE-agent (Yang et al., 2024) designs interfaces for navigating and modifying software repositories. Learning from these interactions also requires a representation in which useful discoveries can accumulate. Reflexion (Shinn et al., 2023) retains verbal reflections on task feedback, while ACE (Zhang et al., 2025b) incrementally develops a playbook of strategies and lessons. Voyager (Wang et al., 2024a) stores executable skills in a growing library, allowing previously learned behaviors to support new tasks. Self-improving systems extend what the agent can revise: the Darwin Gödel Machine (Zhang et al., 2025a) modifies its own agent code, and Prime Agent (Karten et al., August 2026) updates persistent prompts, memories, and skills within a programmable harness.

The interaction record itself can also serve as a durable resource. PRO-LONG (Fox et al., July 2026) preserves a complete, structured history that an agent can query with code; its agents sometimes also construct transition models and search them to plan. VISTA (Han et al., August 2026) combines direct visual observations, free-form language reasoning, and lossless visual memory to support interaction. Schema makes an evolving executable theory the persistent object around which learning and action are organized. The history supplies evidence for checking that theory, and the theory supplies predictions for experiments and plans. This gives accumulated knowledge an operational role: a discovered rule can be tested against earlier observations and used to simulate situations the agent has not yet encountered. The program remains available across context compactions, so later decisions can build directly on earlier discoveries.

A.2 Code World Models

Model-based reinforcement learning uses a model of the environment to support decision-making and learning. Dyna (Sutton, 1990) integrates learning from real experience with updates based on model-generated transitions. MuZero (Schrittwieser et al., 2020) combines a learned dynamics model with tree search, predicting rewards, values, and policies for planning, while DreamerV3 (Hafner et al., 2025) learns behavior through trajectories imagined in a neural world model. Schema shares the use of model predictions to make interaction more productive. It accumulates environment knowledge through revisions to an explicit program while keeping the base language model frozen, making each change available for inspection, replay, and subsequent planning.

Explicit representations of environment mechanisms also have a long history in reinforcement learning. Symbolic approaches learn relational rules describing action effects (Pasula et al., 2007), and

theory-based reinforcement learning induces structured theories of objects, interactions, and termination conditions for exploration and planning (Tsividis et al., 2021). Programs offer a representation in which such knowledge can be inspected, revised, and executed to predict future states.

LLMs make it possible to synthesize these models in general-purpose programming languages. WorldCoder (Tang et al., 2024) learns Python transition and reward functions, using consistency with experience and an optimistic planning criterion to guide program refinement through REx search. CWM (Dainese et al., 2024) synthesizes simulators from offline trajectories, using execution feedback within a Monte Carlo tree search over code generation, improvement, and repair. TheoryCoder (Ahmed et al., 2025) combines symbolic high-level planning with simulation in a learned program; TheoryCoder-2 (Ahmed et al., 2026) further learns reusable abstractions from experience. Other work expands the model class and learning setting. PoE-World (Piriyakulkij et al., 2025) combines programmatic experts into a probabilistic transition model, while One Life to Learn (Khan et al., 2025) studies symbolic modeling of stochastic environments from unguided exploration. Alice (Seo et al., 2026) uses model revisions that explain new transitions but fail on earlier ones to identify missing distinctions and guide exploration.

Schema builds on this foundation at the level of agent architecture: the LLM controls when and how to revise its theory, check it against experience, investigate an uncertain mechanism, or plan toward a goal. This control also covers the program's state grounding, allowing the agent to develop its representation alongside its transition rules and understanding of task completion. Concurrent work on ARC-AGI-3 likewise investigates executable modeling, interaction feedback, and planning (Rodionov, 2026; Courtis et al., 2026; Furgale et al., 2026; Lehmann et al., 2026; Skoutnev et al., 2026). Our study explores this direction as a general approach to agent design across different forms of unfamiliar environments.

A.3 Program Induction and Active Experimentation

Program induction searches for executable explanations of examples, often using a structured program space to make this search tractable. Programming-by-example methods synthesize programs within domain-specific languages; for example, the string-transformation system of Gulwani (2011) infers programs from input–output examples. Sketching (Solar-Lezama et al., 2006; Solar-Lezama, 2008) completes partial programs against a specification, using counterexamples from a verifier to constrain subsequent candidates. DreamCoder (Ellis et al., 2021) learns both a library of reusable program components and a neural search policy, allowing abstractions discovered across tasks to guide later synthesis. These approaches develop complementary ways to construct, check, and reuse programs as representations of knowledge.

Language models provide a further source of candidate hypotheses. Hypothesis Search (Wang et al., 2024b) translates language hypotheses into executable programs and evaluates them on examples. Qiu et al. (2024) similarly study iterative hypothesis refinement, showing how symbolic interpretation helps evaluate and apply rules proposed by language models. Schema places this synthesis-and-checking process within ongoing interaction: recorded transitions provide tests for successive program revisions, and prediction failures supply concrete evidence for further induction. Replaying the full history checks whether a proposed repair also preserves the mechanisms learned earlier.

In an interactive environment, the learner also chooses which evidence to collect. Piriyakulkij et al. (2024) combine language hypotheses with probabilistic inference and information-theoretic experiment design in a rule-discovery task. BoxingGym (Gandhi et al., 2025) studies experimental design and model discovery together in environments defined by generative probabilistic models. Schema connects this process to action in an unfamiliar environment: the agent can use its current program to reach a situation that tests a suspected mechanism, compare the outcome with its prediction, and incorporate the resulting evidence into the same theory used for task planning. Experimentation and task progress therefore draw on a common, continually revised account of how the environment works.

B Harness and Tool Interfaces

Schema gives a frozen language model a persistent workspace and tools for the four operations in Section 3 (Figure 2). Benchmark adapters supply observations, actions, and task feedback; the agent

decides which operation to invoke next. Below, the program contract follows Equation 1, and tools are grouped by function.

Program Workspace. The workspace stores P, notes, helper scripts, and $\mathcal{H}_t$, which the harness extends after every executed action. These artifacts persist across turns, levels, and context compactions. The internal state z_t represents the current situation, including information inferred from past observations. The harness reconstructs it by replaying the relevant history under the current program, with initialization at level entries and resets as defined by the benchmark. Learned rules therefore remain available as the agent grounds them in each new situation.

A turn supplies the current observation, available actions, progress, and remaining budgets where applicable. The agent invokes tools as needed and ends deliberation by committing actions. Execution returns a new observation and a report of the actions taken and any event that interrupted the sequence.

The program defines how observations map to objects, attributes, and relations, and how information needed across successive actions is represented in z_t. Given (z_t, o_t, a_t), the program returns $(\hat{o}_{t+1}, z_{t+1})$ as in Equation 1. The predicted observation includes task feedback such as level completion, failure, or game completion. Recorded observations support replay checks; predicted observations support simulated rollouts. A target predicate $g(z, o)$ specifies a desired outcome: task completion, an intermediate waypoint, or a situation in which an uncertain rule can be tested.

Hypothesize. File tools let the agent read, write, edit, search, and organize its workspace. It can revise the state representation and transition rules, maintain candidate programs, and develop helper scripts for analyzing observations. The `read_history` tool retrieves recorded transitions by level, index, range, or event type, with summaries or detailed before-and-after observations. This lets the agent revisit the evidence for a rule and examine transitions that its current program does not explain.

Certify. The `backtest` tool replays recorded interactions under the program and compares its predictions with observed outcomes. Replay covers the recorded levels by default; a narrower scope helps localize a mismatch. The agent can also test a candidate file before adopting it. Feedback includes prediction accuracy, mismatched indices, and predicted versus observed outcomes. The `model_predict` tool evaluates a selected action from the current observation and reconstructed program state, allowing the agent to inspect a prediction before acting. Replay checks the observations produced during gameplay and the task feedback indicating completion or failure. For ARC-AGI-3, level changes and terminal screens are checked through their corresponding outcome flags. In the color-rotator example of Figure 2, backtesting identifies the unexplained transition and confirms the revised rule against the accumulated history. Abbreviated reports are:

```
backtest [all transitions]:  138/139 transitions fully correct; 1
mismatch
  mismatched transitions (index:kind):  #139:observation
  #139 action=2  recorded before / after, predicted after ...
backtest [all transitions]:  139/139 transitions fully correct; 0
mismatches
```

Plan. The agent selects a target and searches over rollouts of P from the current state, without consuming environment actions (Equation 3). It can use built-in procedures such as BFS and A*, or develop planning code in the workspace. The agent chooses the target, candidate actions, and computational budget, and can supply a heuristic or intermediate waypoint. For ARC-AGI-3, it can select click coordinates and allow a plan to begin with a level reset. A successful search returns an action sequence and search statistics; an unsuccessful search reports whether the explored space was exhausted or a budget was reached. The agent uses this feedback to adjust the search, investigate its model, or commit a plan. For the level in Figure 2:

```
BFS: goal in 15 steps via level_up; expanded 2496 nodes, 795 distinct
states
Plan (-> commit_actions):  [2, 2, 4, 1, 1, 1, 1, 4, 1, 4, 1, 1, 3, 3,
3]
```

Act with verification. The `commit_actions` tool ends deliberation and submits an action sequence, obtained through planning or chosen directly by the agent for exploration. The harness executes committed actions one at a time, records each transition in $\mathcal{H}_t$, and checks the program's prediction against the outcome. Matching predictions allow execution to continue without another LLM call; a mismatch discards the remaining actions and returns the unexpected transition to the agent. Task events such as level completion or failure also return control to the agent. Exploration begins with individual actions whose outcomes provide evidence for building the program. An example interruption report is:

```
world model MISPREDICTED the step just taken (action 2); the rest of
the committed plan was dropped.
```

ARC-AGI-3 observations are grids of hexadecimal color indices, with rendered frames available for inspection; DiG-bench supplies its native text observations; MazeBench supplies an ASCII rendering of the current room. Each adapter exposes the benchmark's native actions and task feedback, allowing the same operations to support program revision, replay, planning, and verified execution across the three environments.

C Benchmarks and Backgrounds

We describe the three benchmarks below and summarize their published model results in Tables 2–4.

C.1 ARC-AGI-3

ARC-AGI-3 (ARC Prize Foundation, 2026a) presents visual games as 64×64 grids, with mechanics and objectives that must be discovered through interaction. Successive levels recombine these mechanics into increasingly difficult puzzles, testing whether an agent can turn its discoveries into efficient plans. The benchmark contains 25 public games, 55 semi-private games for organizer-run evaluations, and 55 private competition games. The latter two sets are not released for general academic use; we evaluate all 25 public games, comprising 183 levels (Figure 10). The ablations in Section 4.1 use the ten games with the most actions in their human baselines, marked in Table 6; together they contain 75 levels.

Relative human action efficiency (RHAE) combines level completion with action efficiency relative to first-time human play. For a game with L levels of which the agent completes the first k, the score is

$$\mathrm{RHAE} = \min\left(\frac{\sum_{l=1}^{k} l}{\sum_{l=1}^{L} l},\ \frac{\sum_{l=1}^{k} l \cdot \min\big(1.15,\ (h_l/a_l)^2\big)}{\sum_{l=1}^{L} l}\right), \tag{4}$$

where h_l and a_l are the human baseline's and agent's action counts on level l. Later levels receive greater weight, and exceeding the human action count incurs a quadratic penalty. We report the mean score over games as a percentage.

C.2 DiG-bench

DiG-bench (Battleday et al., 2026a) comprises 70 text games across seven difficulty tiers, with one to sixteen levels per game. Observations are strings of letters, numbers, and symbols, and actions are individual characters; both transition rules and win conditions must be inferred. Limited lives and per-level step budgets make informative experiments essential: a game is won only when all levels are cleared before these budgets run out. Many games also provide a creative mode for testing hypotheses outside the main level (Battleday et al., 2026b). Three games per tier are public, while the remaining 49 are held privately for evaluation; our experiments cover all 21 public games.

C.3 MazeBench

MazeBench (Pappas & Pappas, 2026a) is a three-dimensional exploration and puzzle environment with 256 interconnected rooms and 100 gems (Figure 11). Progress requires discovering mechanisms involving pushable blocks, lifts, bridges, and icy surfaces, then applying them across distant

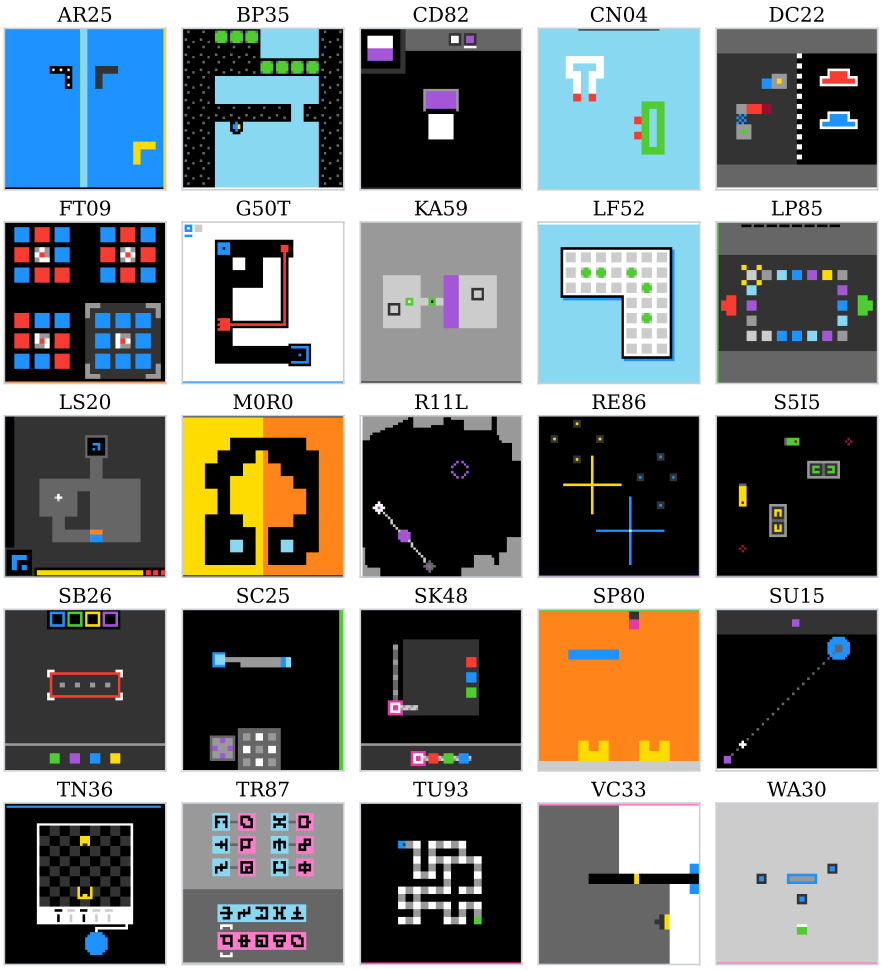


Figure 10: The 25 public ARC-AGI-3 games, each shown at the start of its first level.

rooms. We use the ASCII track, which renders the current room from a rotatable camera. The interface exposes eleven actions for movement, camera control, undo, room reset, and return to visited rooms (Pappas & Pappas, 2026b). Gems collected and rooms visited measure puzzle-solving and exploration over long interactions, making this environment useful for studying how agents accumulate and reuse knowledge.

Table 2: Frontier models on ARC-AGI-3, from ARC Prize's reports and leaderboard as of September 2026 (ARC Prize Foundation, 2026a;e;d;g;f). Model dates follow the cited vendor documentation; empty cells are unreported. †Approximate public score for Fable-class models (ARC Prize Foundation, 2026c). §Parentheses show the provider-adapter harness (ARC Prize Foundation, 2026b).

			RHAE (%)	
Model	Cutoff	Release	Public	Semi-private
Gemini 3.1 Pro (Google DeepMind, 2026)	Jan 2025	Feb 2026		0.4
GPT-5.4 (OpenAI, 2026b)	Aug 2025	Mar 2026		0.2
Grok 4.20 (xAI, 2026)	Sep 2025	Mar 2026		0.1
Claude Opus 4.7 (Anthropic, 2026d)	Jan 2026	Apr 2026		0.2
GPT-5.5 (OpenAI, 2026c)	Dec 2025	Apr 2026		0.4
Claude Opus 4.8 (Anthropic, 2026e)	Jan 2026	May 2026		1.5
Claude Fable 5 (Anthropic, 2026b)	Jan 2026	Jun 2026	20†	
Grok 4.5 (xAI, 2026)	Feb 2026	Jul 2026	0.3	
GPT-5.6 Luna (OpenAI, 2026d)	Feb 2026	Jul 2026	0.0	0.2
GPT-5.6 Terra (OpenAI, 2026d)	Feb 2026	Jul 2026	2.3	0.8
GPT-5.6 Sol (OpenAI, 2026d)	Feb 2026	Jul 2026	13.3	7.8
Grok 4.6 (xAI, 2026)	Feb 2026	Aug 2026		2.1
Human (ARC Prize Foundation, 2026a)	Mar 2026	—	100	100
Claude Opus 5 (Anthropic, 2026f)	May 2026	Jul 2026		30.2
GPT-6 Astra (OpenAI, 2026e)	Apr 2026	Sep 2026		62.7 (99.9)§

Table 3: Frontier models in the basic harness on all 70 DiG-bench games, as reported by the benchmark authors (Battleday et al., 2026a). Win rates average runs within each game, then games within each set; tiers 6–7 contain 20 games. Each game was solved by at least one first-time human player.

			Win rate (%)	
Model	Cutoff	Release	All tiers	Tiers 6–7
Gemini 3.1 Pro (Google DeepMind, 2026)	Jan 2025	Feb 2026	16.7	0.0
Qwen 3.6 27B (Qwen Team, 2026)	Oct 2025	Apr 2026	1.4	0.0
GPT-5.5 (OpenAI, 2026c)	Dec 2025	Apr 2026	25.7	10.0
GLM-5.2 (Z.ai, 2026)	Mar 2026	Jun 2026	15.3	0.0
Kimi K3 (Moonshot AI, 2026)		Jul 2026	22.9	0.0
Claude Opus 5 (Anthropic, 2026f)	May 2026	Jul 2026	71.4	40.0
DeepSeek V4 Flash (DeepSeek-AI, 2026)		Jul 2026	11.4	0.0
DeepSeek V4 Pro (DeepSeek-AI, 2026)	Apr 2026	Aug 2026	4.3	0.0
Human solvability (Battleday et al., 2026a)	—	—	100	100

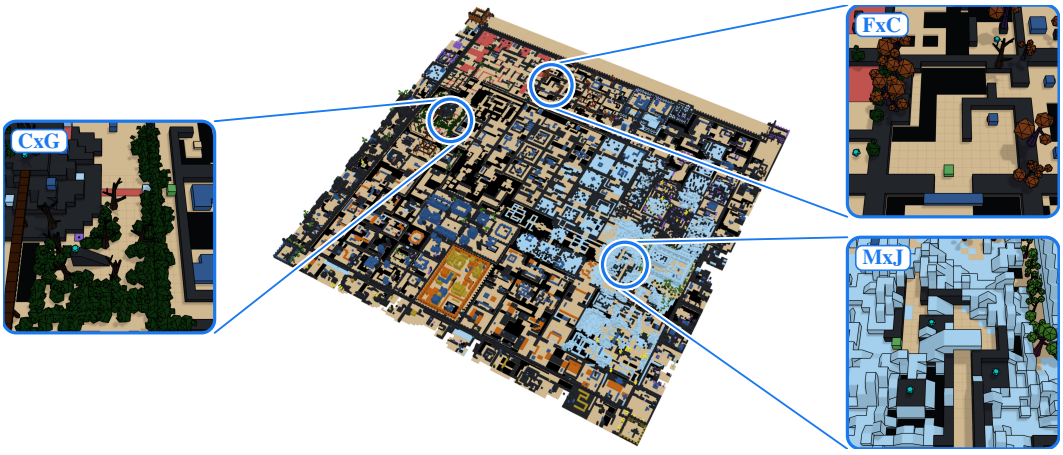


Figure 11: **The MazeBench environment.** The full world of 256 interconnected rooms, with enlarged views of rooms CxG, FxC, and MxJ. Blue circles and connecting lines locate each room within the world.

Table 4: Frontier models on MazeBench's ASCII track, from the September 14, 2026 leaderboard (Pappas & Pappas, 2026a). Tools permit Python execution. Gem percentages use each run's available total (71–100); rooms are out of 256. The human row gives the top-50 median, with gems out of 100; individual entries appear in Table 5.

			With tools		Without tools	
Model	Cutoff	Release	Gems (%)	Rooms	Gems (%)	Rooms
Gemini 3.1 Pro (Google DeepMind, 2026)	Jan 2025	Feb 2026	1.1	5	0.0	1
GPT-5.4 (OpenAI, 2026b)	Aug 2025	Mar 2026			0.0	2
GPT-5.5 (OpenAI, 2026c)	Dec 2025	Apr 2026			0.0	4
Claude Opus 4.8 (Anthropic, 2026e)	Jan 2026	May 2026			0.0	4
GLM-5.2 (Z.ai, 2026)	Mar 2026	Jun 2026			0.0	1
Claude Fable 5 (Anthropic, 2026b)	Jan 2026	Jun 2026	14.7	43	1.4	10
GPT-5.6 Luna (OpenAI, 2026d)	Feb 2026	Jul 2026	0.0	2	0.0	1
GPT-5.6 Terra (OpenAI, 2026d)	Feb 2026	Jul 2026	6.7	12	0.0	2
GPT-5.6 Sol (OpenAI, 2026d)	Feb 2026	Jul 2026	17.1	67	1.4	4
Kimi K3 (Moonshot AI, 2026)		Jul 2026	1.3	4	0.0	2
Claude Opus 5 (Anthropic, 2026f)	May 2026	Jul 2026	16.0	62	1.3	8
DeepSeek V4 Pro (DeepSeek-AI, 2026)	Apr 2026	Aug 2026	0.0	2	0.0	1
Grok 4.6 (xAI, 2026)	Feb 2026	Aug 2026	1.1	6		
Claude Fable 5.1 (Anthropic, 2026c)	Jun 2026	Sep 2026	11.1	33	2.0	12
GPT-6 Astra (OpenAI, 2026e)	Apr 2026	Sep 2026	23.0	113	14.0	103
Human top-50 (Pappas & Pappas, 2026a)	—	—			29.0	137

Table 5: Top-50 human entries used for the MazeBench reference, comprising submissions through September 19, 2026 (Pappas & Pappas, 2026a). Names are omitted; gems and rooms are out of 100 and 256, respectively, and hours denote recorded playtime.

Rank	Gems	Rooms	Actions	Hours	Rank	Gems	Rooms	Actions	Hours
1	80	233	115,060	55.99	26	29	104	49,221	95.95
2	80	233	151,663	27.21	27	26	139	40,112	12.31
3	80	233	260,784	102.35	28	26	126	68,310	13.64
4	79	233	63,543	10.82	29	21	174	27,181	22.36
5	79	233	206,116	29.43	30	21	129	26,261	5.04
6	79	233	221,343	72.30	31	21	105	31,332	11.62
7	79	233	227,236	286.17	32	21	104	31,654	9.79
8	77	233	169,758	99.46	33	20	72	21,845	23.48
9	68	209	119,655	23.30	34	17	96	5,001	21.48
10	60	210	160,616	77.97	35	16	70	25,278	2.80
11	59	233	120,523	80.42	36	15	62	21,428	24.80
12	55	199	141,337	76.70	37	14	96	30,209	5.78
13	49	202	54,560	11.56	38	13	105	31,618	7.97
14	49	173	47,275	51.49	39	13	42	9,311	4.76
15	45	174	42,721	26.39	40	12	62	40,629	5.63
16	45	155	62,784	225.78	41	10	33	11,231	2.93
17	40	162	94,565	23.07	42	9	49	27,759	8.52
18	38	171	66,138	15.01	43	9	47	8,299	9.74
19	38	156	39,231	72.14	44	9	20	10,299	1.59
20	37	153	34,971	9.78	45	8	104	23,064	10.40
21	36	165	62,131	21.71	46	8	86	11,620	13.85
22	33	135	51,825	15.06	47	8	65	4,734	4.94
23	32	164	58,978	31.11	48	8	27	11,690	12.61
24	30	209	43,640	61.86	49	7	57	38,219	9.04
25	29	119	64,195	9.42	50	7	41	6,834	0.89

D ARC-AGI-3 PER-GAME RESULTS

Table 6 reports every public game for the four base models, and Figures 12–15 trace each run level by level against the human baseline. With Claude Fable 5, Schema clears all 183 levels in 57% of the human baseline's actions and uses fewer actions than the human on 163 of them; with GPT-5.6 Sol at maximum effort it also clears every level, in 63% of the human actions.

E DIG-BENCH REPEATED RUNS

We ran every DiG-bench configuration three times with GPT-6 Astra. Table 7 gives the mean and standard deviation of the games won and the levels completed, and Figures 16 and 17 break them down by reasoning effort, difficulty tier, and game. Schema wins more games than the basic harness in every run at every reasoning effort, and its advantage is largest at medium effort, where it wins 18.7 games on average against 10.7; at medium effort it already matches the basic harness at maximum effort. The gains come from the hardest games: across all efforts and runs, Schema wins 159 of the 162 game runs in tiers 1–6 and 16 of the 27 in tier 7, against 4 for the basic harness.

Table 6: Per-game ARC-AGI-3 results of Schema with each base model. *Levels* is the number of levels in the game and *Human* the total actions of the human baseline; for each model, *RHAE* is the game's score and *Actions* counts actions through the last cleared level. Parentheses give completion counts for partially cleared games. *The ten games with the most human actions, used for the ablations in Section 4.1.

			Opus 4.8		Fable 5		Sol xhigh		Sol max	
Game	Levels	Human	RHAE	Actions	RHAE	Actions	RHAE	Actions	RHAE	Actions
AR25*	8	748	100.0	269	100.0	298	100.0	261	100.0	278
BP35	9	651	62.9	1,265	93.5	566	28.8	218 (5/9)	60.9	1,347
CD82	6	171	100.0	121	100.0	144	100.0	116	100.0	86
CN04*	6	789	100.0	479	100.0	324	100.0	318	100.0	241
DC22*	6	1,228	38.9	485 (4/6)	98.7	1,205	100.0	1,018	100.0	814
FT09	6	208	100.0	94	100.0	78	100.0	97	100.0	92
G50T*	7	879	100.0	544	96.4	486	100.0	457	100.0	306
KA59	7	730	100.0	431	100.0	436	100.0	414	100.0	430
LF52*	10	1,339	23.8	475 (5/10)	100.0	1,030	100.0	875	100.0	844
LP85	8	388	100.0	134	100.0	99	100.0	120	100.0	98
LS20*	7	776	100.0	642	100.0	497	100.0	398	100.0	479
M0R0*	6	1,107	100.0	221	100.0	278	100.0	271	100.0	242
R11L	6	233	100.0	83	100.0	111	100.0	90	100.0	85
RE86*	8	1,255	100.0	615	100.0	635	100.0	822	100.0	742
S5I5	8	638	89.9	643	100.0	294	100.0	356	100.0	318
SB26	8	213	82.5	347	98.6	135	100.0	127	100.0	131
SC25	6	350	93.9	363	100.0	334	42.5	1,285	82.7	359
SK48*	8	1,070	85.1	801	100.0	443	63.9	1,402	87.8	977
SP80	6	518	56.3	450 (5/6)	100.0	283	100.0	301	100.0	164
SU15	9	361	61.5	444	100.0	158	80.3	593	100.0	160
TN36	7	317	75.3	348	94.7	210	80.3	584	87.0	900
TR87	6	414	100.0	138	100.0	174	100.0	208	100.0	155
TU93	9	462	100.0	243	100.0	195	100.0	255	100.0	246
VC33	7	447	81.8	507	99.1	342	100.0	261	100.0	240
WA30*	9	1,843	100.0	956	100.0	1,080	100.0	1,403	100.0	980
Mean / total	183	17,135	86.1	11,098	99.2	9,835	91.8	12,250	96.7	10,714
Games won			22/25		25/25		24/25		25/25	

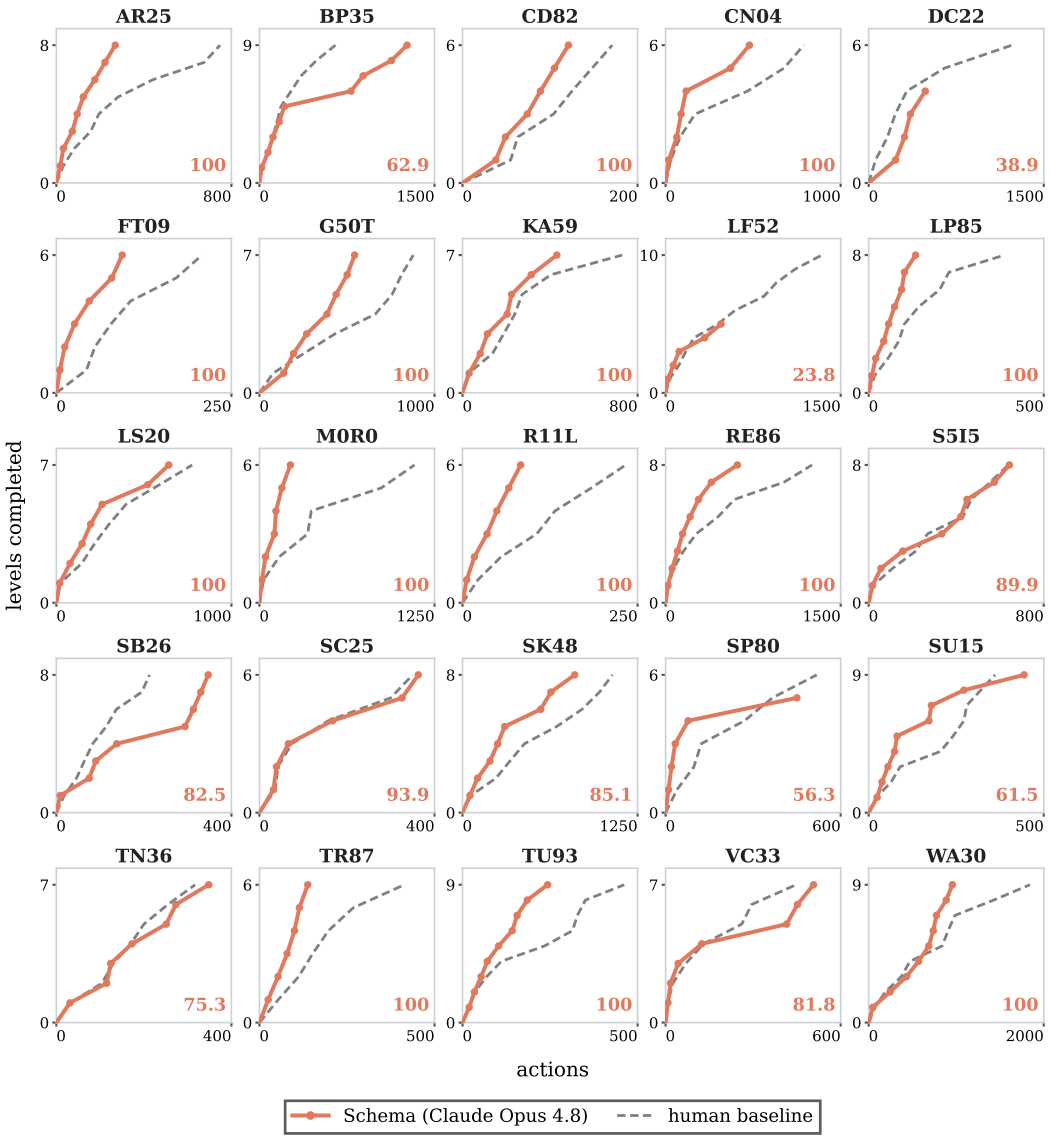


Figure 12: Per-game progress on ARC-AGI-3 with Claude Opus 4.8 in Schema. Curves connect cumulative action counts at level completion for Schema and the human baseline. The number in each panel is the game's RHAE.

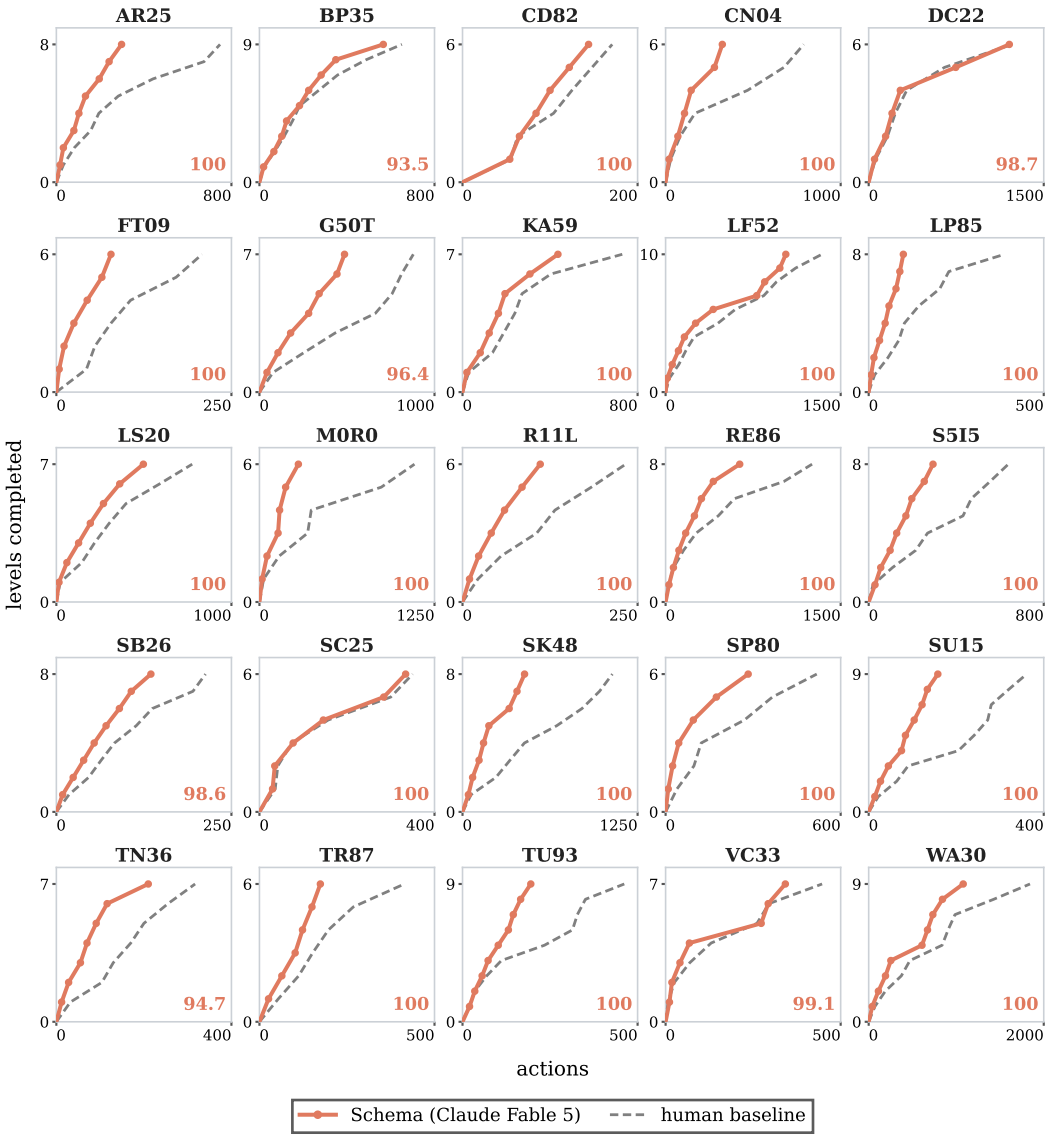


Figure 13: Per-game progress on ARC-AGI-3 with Claude Fable 5 in Schema, as in Figure 12.

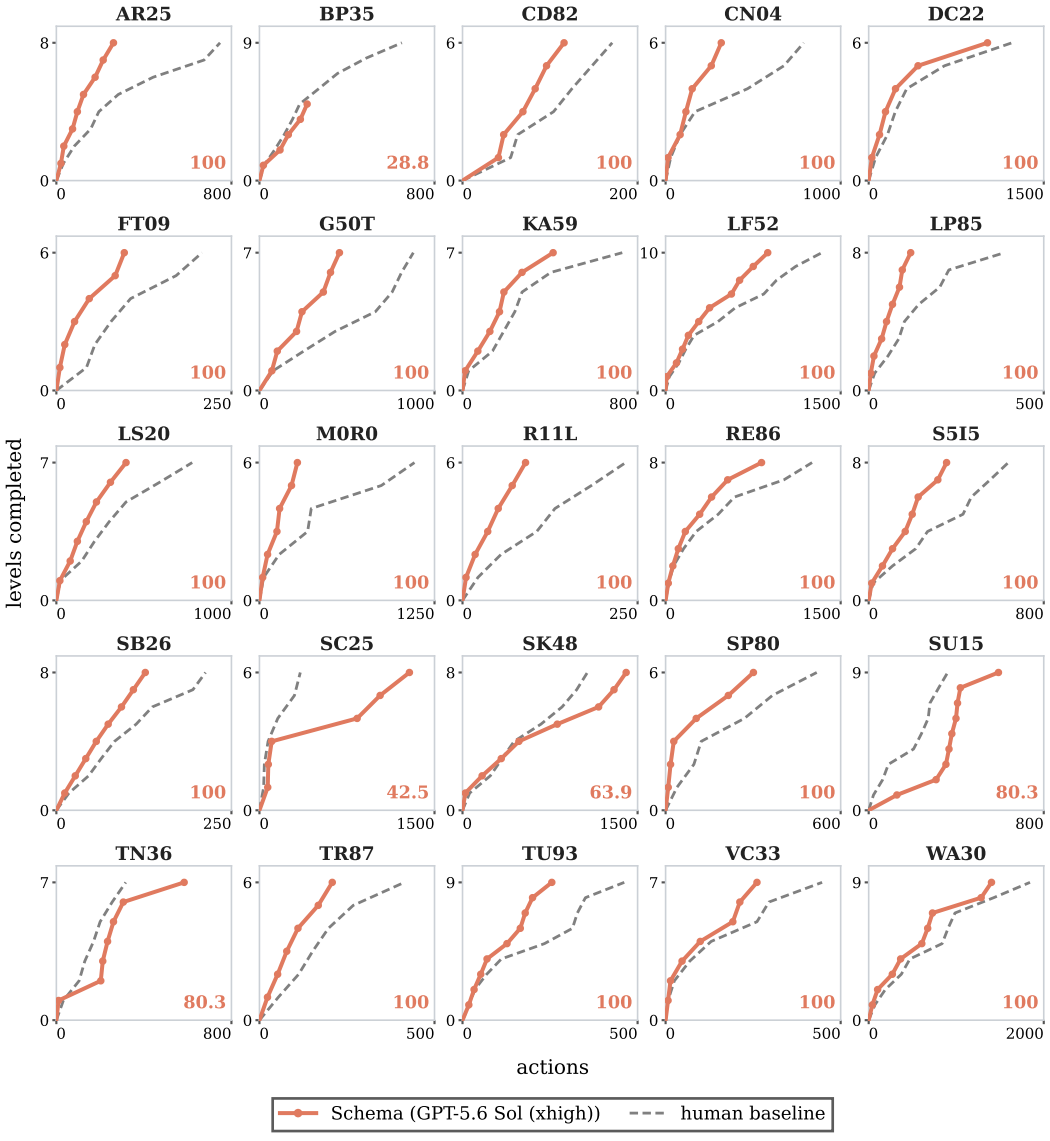


Figure 14: Per-game progress on ARC-AGI-3 with GPT-5.6 Sol at extra-high effort in Schema, as in Figure 12.

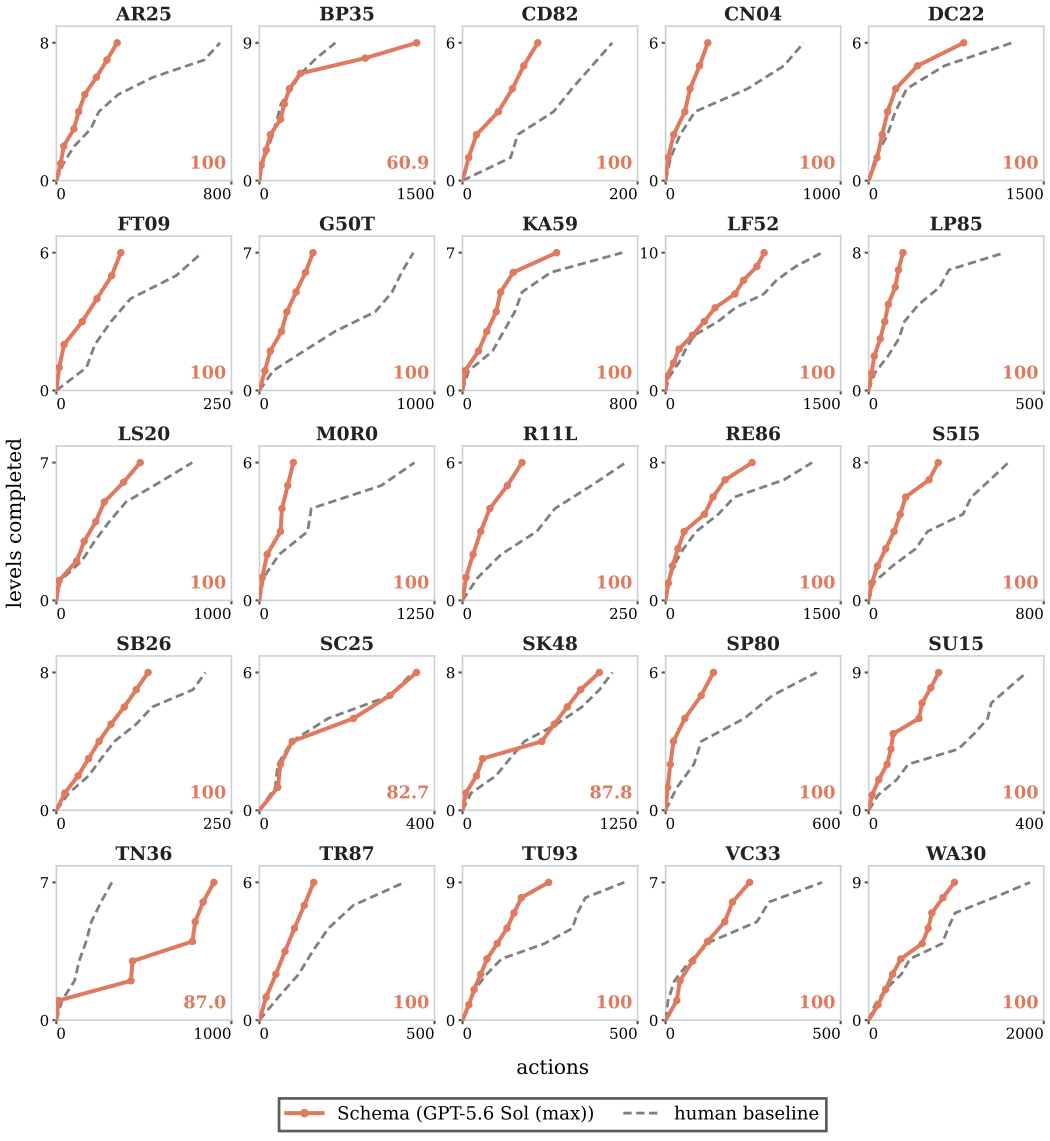


Figure 15: Per-game progress on ARC-AGI-3 with GPT-5.6 Sol at maximum effort in Schema, as in Figure 12.

Table 7: DiG-bench with GPT-6 Astra over three runs: mean ± standard deviation of the games won (of 21) and the levels completed (of 193).

		Medium	High	Max
Games won	Basic harness	10.7 ± 0.6	16.0 ± 1.0	18.7 ± 0.6
	Schema	**18.7 ± 1.5**	**19.3 ± 1.2**	**20.3 ± 0.6**
Levels completed	Basic harness	126.7 ± 7.8	159.3 ± 0.6	179.3 ± 5.0
	Schema	**177.7 ± 7.0**	**183.7 ± 2.3**	**187.7 ± 4.6**

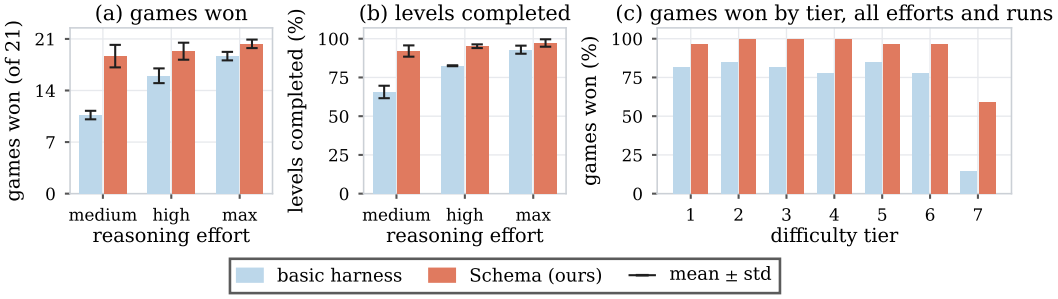


Figure 16: DiG-bench with GPT-6 Astra over three runs. (a) Games won and (b) levels completed at each reasoning effort: bars show the mean and error bars one standard deviation. (c) Share of games won in each difficulty tier, over all efforts and runs.

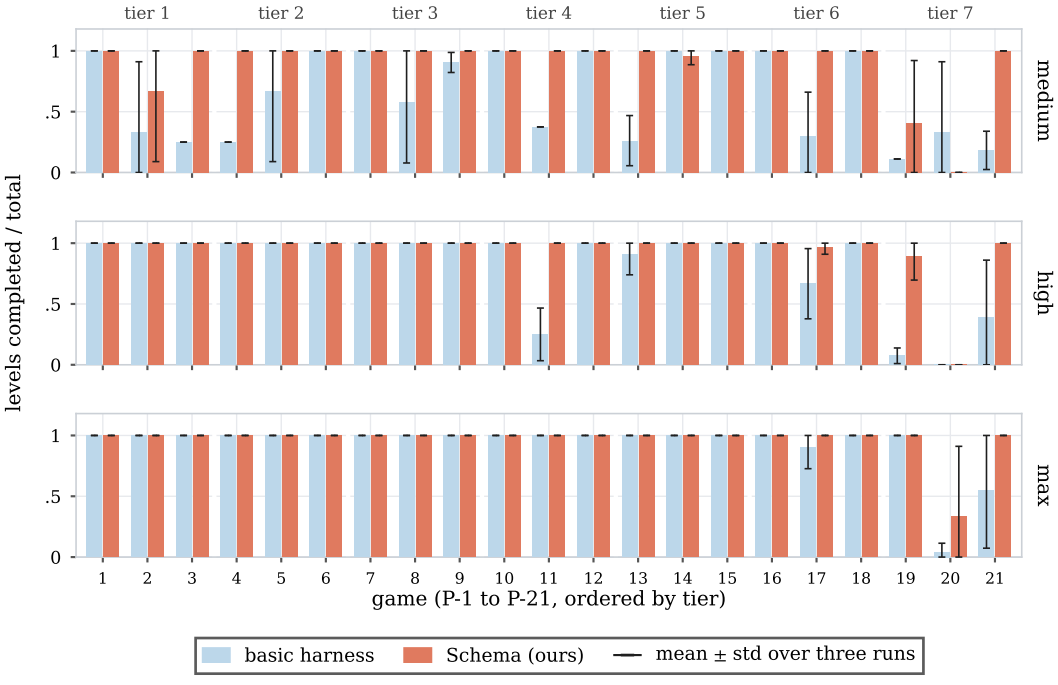


Figure 17: Fraction of levels completed in each DiG-bench game at each reasoning effort, mean and standard deviation over three runs, games ordered by tier.

F MazeBench: Gem Collection and Room Coverage

Figure 18 maps room coverage for the four trajectories in Figure 8. Schema and Codex with GPT-6 Astra share 106 visited rooms: 33 additional rooms appear only in Schema's trajectory and seven only in Codex's. Schema also reaches every room visited by each of the other two baselines.

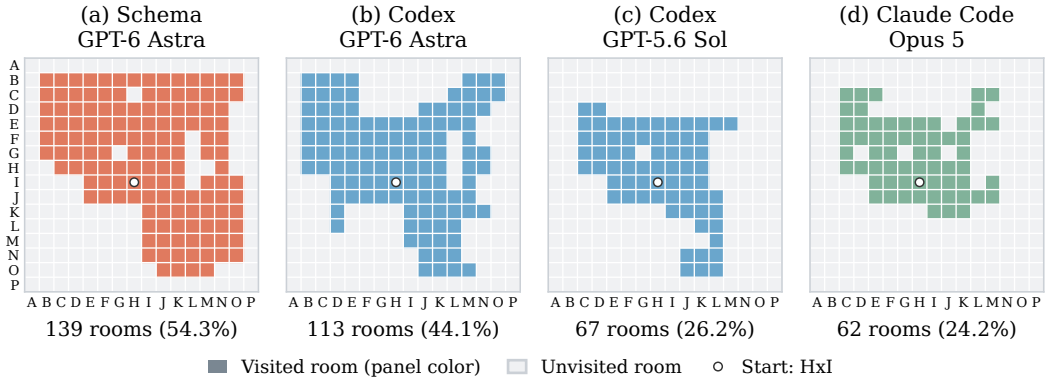


Figure 18: **Room coverage of the four MazeBench runs.** Each cell is a room in the same 16×16 grid, with columns and rows A–P (e.g., CxG is column C, row G). Panels show each run's recorded endpoint. Baselines use the September 14, 2026 ASCII tools leaderboard, as in Figure 8.

Table 8 lists all 33 gems, collected across 31 rooms. Coordinates are recovered from recorded top views and the gem's disappearance at collection. Steps count all issued actions, including camera changes, undo, reset, and room returns. Counts start at one: raw history index i corresponds to step $i + 1$, so the final index 27,818 represents 27,819 actions.

Table 8: **All 33 gems collected by Schema.** Read down the left block, then the right. Room-local (x, y) coordinates range from 0 to 15 in the unrotated top view, increasing rightward and downward. Step is the cumulative action count; Δ counts actions since the previous gem (from the start for gem 1), including exploration elsewhere.

Gem	Room	(x, y)	Step	Δ	Gem	Room	(x, y)	Step	Δ
1	HxH	$(1, 3)$	100	100	18	ExF	$(2, 6)$	11,173	730
2	GxH	$(10, 1)$	303	203	19	ExC	$(13, 12)$	11,478	305
3	HxF	$(1, 1)$	512	209	20	DxG	$(14, 2)$	12,469	991
4	GxF	$(2, 9)$	749	237	21	IxJ	$(7, 14)$	12,643	174
5	FxE	$(11, 2)$	1,001	252	22	JxL	$(14, 14)$	13,759	1,116
6	FxF	$(10, 2)$	1,116	115	23	IxE	$(1, 10)$	15,700	1,941
7	IxI	$(1, 13)$	2,731	1,615	24	MxN	$(1, 15)$	17,168	1,468
8	JxG	$(12, 8)$	3,170	439	25	NxN	$(8, 8)$	17,815	647
9	JxE	$(1, 1)$	3,463	293	26	MxJ	$(3, 13)$	18,779	964
10	MxD	$(12, 5)$	4,163	700	27	MxJ	$(12, 11)$	19,535	756
11	JxH	$(4, 5)$	4,425	262	28	MxJ	$(5, 5)$	19,554	19
12	GxI	$(4, 12)$	4,744	319	29	MxI	$(11, 5)$	19,742	188
13	CxE	$(2, 3)$	5,019	275	30	OxK	$(4, 11)$	20,247	505
14	DxH	$(6, 5)$	5,182	163	31	MxL	$(14, 8)$	21,732	1,485
15	CxD	$(3, 12)$	6,042	860	32	FxB	$(13, 13)$	26,497	4,765
16	FxC	$(0, 4)$	7,244	1,202	33	NxE	$(11, 14)$	27,528	1,031
17	NxF	$(13, 11)$	10,443	3,199					

G ARC-AGI-3: Cases Behind the Ablations

The following cases expand the examples in Section 4.1 using recorded interactions and program revisions from the Claude Opus 4.8 runs.

Program representation: assembling changing shapes (CN04, level 5). The agent moves and rotates pieces until their connectors meet, while keeping their bodies from overlapping. One piece grows when acted on, changing both its outline and its connectors. In the final search, Schema represents bodies and connectors as separate sets of grid cells and enumerates rotations and translations. After four growth actions, the growing piece has seven connectors; the computed arrangement pairs all fourteen connectors into seven joints without any body overlap (Figure 19). The run clears the level in 279 actions, compared with 1,219 for the prose-model variant.

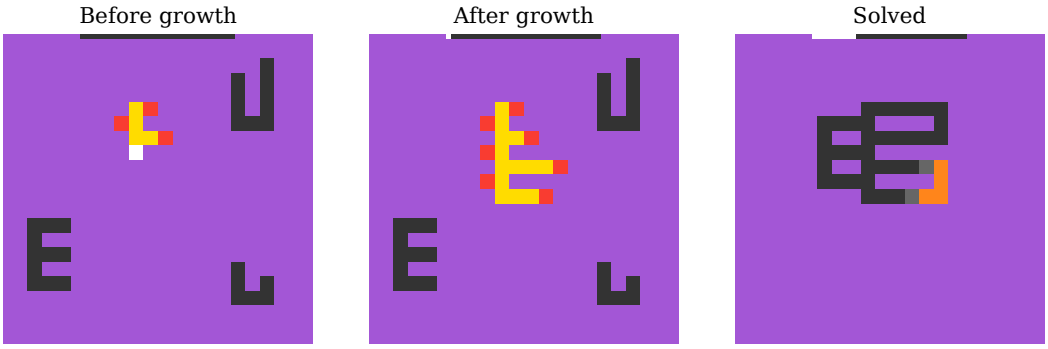


Figure 19: Recorded CN04 frames before growth, after four growth actions, and at level completion. The final arrangement joins the pieces without overlapping their bodies.

Certification: a new rule can break an old prediction (AR25). AR25 requires moving shapes and mirror axes so that the shapes and their reflections cover the targets. To predict an arrow-key action, the program must identify which object is currently selected. In the run without certification, a revision on level 3 addresses a piece whose selection marker is obscured by a target. But the revised inference also changes predictions for the already completed first level: agreement on the same sixteen recorded transitions falls from 16/16 to 1/16. Across the history available at the two checkpoints, agreement falls from 42/43 (97.7%) to 30/46 (65.2%). Further revisions also introduce an error in the reflection geometry: moving a piece one cell left should move its reflection one cell right, but the later program moves the reflection two cells (Figure 20). With certification, Schema's final model reproduces 267/268 transitions (99.6%).

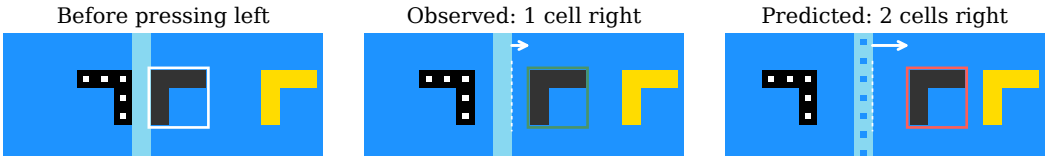


Figure 20: The same historical AR25 transition, shown with identical crops. Boxes mark the reflection; dashed lines mark its original position. The turn-15 program predicts the observed one-cell movement exactly, while the turn-39 program predicts two cells.

Planning: coordinating transformations and movement (LS20, level 5). The agent carries a symbol that must match the goal in both shape and color. Contact with a moving marker rotates the symbol, so reaching the right location is insufficient: the player and marker must meet at the right time, with enough energy left to reach the goal. The variant without planning enters this level with a program that passes all 252 historical transition checks, but works out its routes and transformations manually and takes 233 actions to finish. Schema finishes in 65 actions. Its final thirteen-action plan begins with a down–up pair that makes the player and moving marker converge, rotating the carried symbol into the target shape; the remaining eleven actions reach the goal (Figure 21).

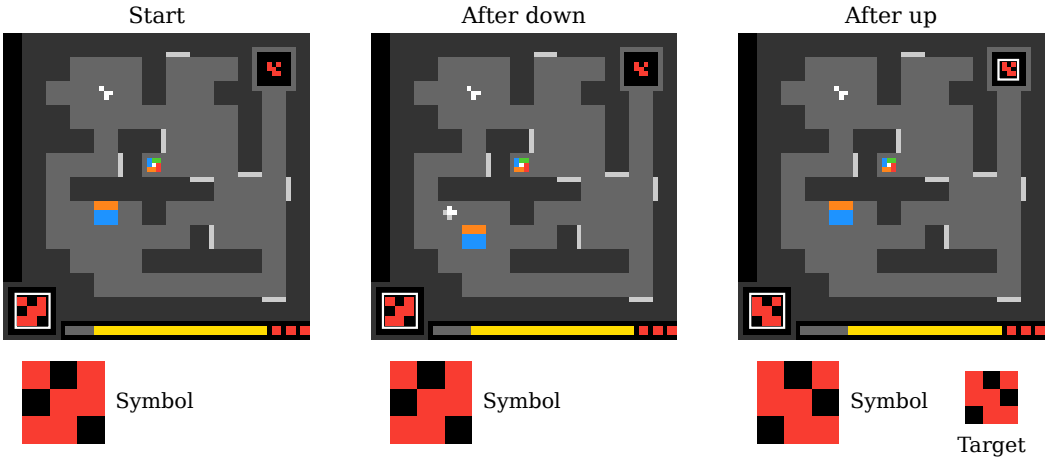


Figure 21: Recorded LS20 observations before and after the first two actions of the final plan. Enlarged crops show the carried symbol rotating to match the target after the down–up pair. The remaining eleven actions reach the goal.

Verified execution: noticing a spent resource (LS20, level 2). Here the agent must rotate its symbol and reach the goal while replenishing a limited energy supply. The run without verification learns that yellow stations refill energy, but initially treats them as reusable. It commits a 43-action plan that relies on returning to a previously used station. The very first action already exposes the mistake: after the player leaves a station, the model predicts that the station remains, while the observation shows bare floor (Figure 22). Execution continues for all 42 remaining actions, including the return to the spent station, and the energy supply is exhausted before the batch ends. Only afterward does the agent revise the program to consume a station after use and require a still-present station for refueling.

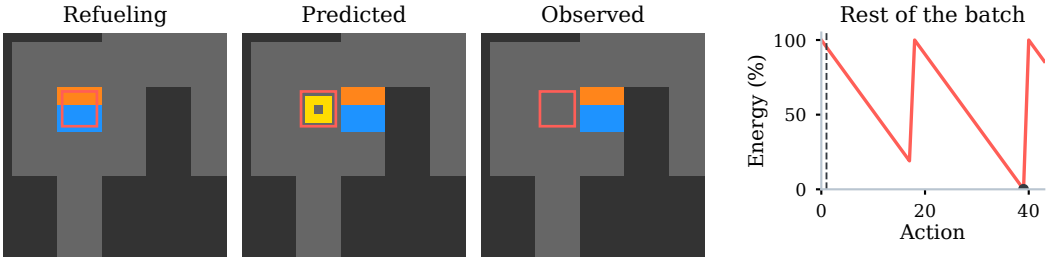


Figure 22: The first prediction mismatch in the 43-action LS20 batch without verification. Identical crops highlight the station's location: the prediction retains it, but the observation shows floor. Execution continues through another refill and eventual energy depletion; the rebound after zero is an automatic reset.

H DiG-bench: Discovering a Hidden String Rule

The puzzle. P-3 asks the agent to discover which strings over `a`, `b`, `c`, and `d` satisfy a hidden rule. Practice queries return yes or no without costing a life; the agent then takes an eight-question quiz, where each wrong classification costs a life. We follow level 3 of a Schema run with GPT-6 Astra at medium effort.

From letter positions to letter counts. After `a` is accepted, the agent first encodes the hypothesis that accepted strings end in `a`. The rejection of `ba` contradicts this rule, so it adds the requirement that the string also start with `a`. The next queries expose the problem with that explanation: `aba` and `ac` are both accepted, although only the former ends in `a`. The agent replaces the positional rule with `s.count('a') > s.count('b')`. It then tests `baa`, `abba`, and `cad`: rearranging the letters preserves acceptance, adding a second `b` removes it, and surrounding `a` with other letters preserves it (Table 9).

Current hypothesis	New practice queries	Consequence
Initial probe	`a`: yes	Propose a rule involving `a`.
Ends in `a`	`b`, `c`, `d`, `ab`, `ba`: no	`ba` rules out the current hypothesis.
Starts and ends in `a`	`aa`, `aba`, `ac`: yes	`ac` rules out this positional rule.
More `a`'s than `b`'s	`baa`: yes; `abba`: no; `cad`: yes	All three agree with the count rule.

Table 9: All twelve practice queries in P-3 level 3, grouped chronologically by the program hypothesis in use. Hypothesis descriptions summarize the recorded code edits.

Checking the rule and taking the quiz. Each revision is checked against the accumulated interaction history. After the final practice queries, the program reproduces all 107 checkable transitions collected so far, including the earlier levels and the string-entry mechanics. Schema then classifies all eight quiz strings correctly without losing a life (Table 10). Across the full game, this run clears all eight levels with one life lost; the basic harness at medium effort stops at level 3 in each of our three runs.

String	`dca`	`a`	`dbdacb`	`bdcaaa`	`dcbdba`	`acd`	`bcabca`	`adbcc`
#a − #b	1	1	−1	2	−1	1	0	0
Answer	yes	yes	no	yes	no	yes	no	no
Correct	✓	✓	✓	✓	✓	✓	✓	✓

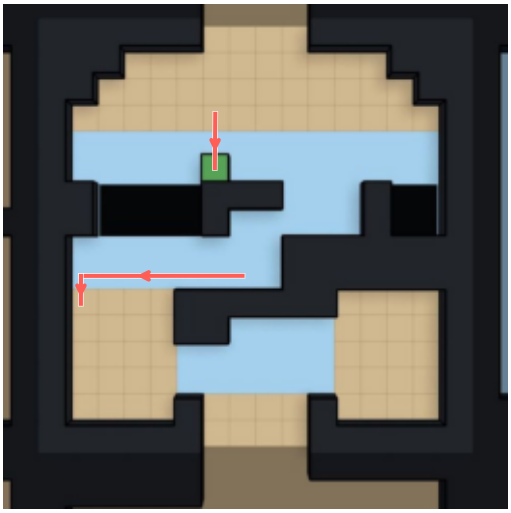
Table 10: The eight scored questions in order. The count difference is shown to make the learned rule easy to check.

I MAZEBENCH: REUSING A RULE AFTER LONG INTERACTION

The puzzle. On ice, one directional action can carry the player across several tiles. The player stops when blocked or upon reaching ordinary ground, so solving an ice maze requires planning a sequence of stopping points rather than simply tracing a walkable path to the gem. We follow the same Schema run with GPT-6 Astra from its first encounter with ice in room IxG to a later maze in room NxE.

Learning a reusable movement rule. At action 2,629, a downward move in IxG slides the player across two tiles and stops just before a wall. The agent adds sliding to its movement program and tests further directions. Nine subsequent probes all match its predictions, including a slide that ends on the first ordinary tile beyond the ice (Figure 23a). By action 2,638, the revised program reproduces all 2,638 recorded transitions.

(a) IxG: discovery, action 2,629



(b) NxE: reuse, action 27,490

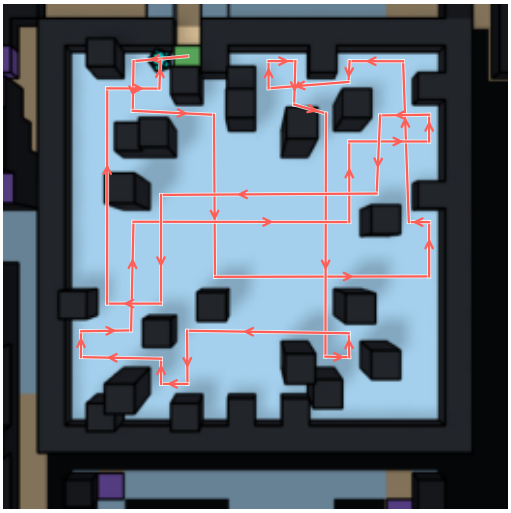


Figure 23: Ice-rule discovery and reuse, shown on top-down game-engine views. Arrows trace selected probes in IxG (left), which stop at a wall or upon leaving ice, and the complete 38-action route to the gem in NxE (right).

Applying the rule in a new room. Schema first enters NxE at action 27,486. From several viewpoints and an initial movement, it recovers the wall and ice geometry, including the ordinary entry tile initially hidden by the player. Comparing the program immediately before entering NxE with the version used for the final plan shows that all 58 top-level functions and classes are unchanged: the additions import and register the new room's geometry. The existing movement code supplies the consequences of sliding through this new layout.

Planning after repeated context compaction. At action 27,490, A* search finds a 38-action route to the gem, 24,852 actions after the IxG probes (Figure 23b). The saved session records at least 35 context compactions between discovery and this plan. All 38 actions execute, and each complete observation matches its saved forward prediction; the last action collects the gem at action 27,528.