RISE: CO-SYNTHESIZING VISUAL GAMES AND OBSERVATION-GROUNDED DEMONSTRATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon visual tasks can require agents to learn how actions affect an environment and apply that knowledge in later decisions. Demonstrations generated by reading a simulator's internal rules can skip the interactions through which an agent would learn how the environment works. We introduce RISE (Rule Inference from Synthesized Environments), a framework that jointly generates executable visual games, connected tasks, and demonstrations based on public observations. Tasks are organized into multi-level sequences that share the same underlying game mechanics. Reusable symbolic solvers collect demonstrations through public images and interaction feedback, without per-step LLM teacher calls. For compositional puzzles, RISE uses observations from early interactions to construct later goals. Their verified solution paths include actions not yet tried in a given state, with outcomes predicted from the recorded observations. Task constructors check solvability, while replay checks link structured evidence records to their source observations and verify executed outcomes. Construction audits across nine game types test execution, evidence provenance, and export consistency. In a separate sixteen-game development study using an earlier generator, synthesized supervision improves task execution, and a revised data recipe outperforms the standard recipe at a similar target-token budget.

1 INTRODUCTION

Long-horizon visual tasks can require an agent to learn how the environment responds to its actions and use that knowledge in later decisions. In a visual game, for example, an agent may test a furnace in one level and use its measured fuel cost to plan production in another. The useful knowledge comes from the change it observes after acting. Training demonstrations for such tasks should show how this knowledge is acquired and where it affects subsequent choices.

A successful demonstration can leave out how the agent obtains the knowledge needed to choose its actions. A solver that reads the simulator's internal rules can choose successful actions without first discovering those rules through interaction. For example, if a planner already knows the furnace cost, its demonstration can skip the experiment that would reveal it. This difference in available information matters when demonstrations train an agent that sees only images and feedback (Chen et al., 2020; Warrington et al., 2021).

Generating environments and demonstrations together provides a way to design where such information is acquired and used. AgentGen constructs environments and tasks with planner-generated trajectories, while SPADE and Echoverse couple environment development with agent interaction (Hu et al., 2025b; Liu et al., 2026; Pandya et al., 2026). We ask how to construct later tasks so that observations from earlier demonstrations support the decisions needed to solve them.

We introduce RISE, *Rule Inference from Synthesized Environments*, which jointly generates executable visual games, matching demonstrations, and connected sequences of tasks. Each game type provides a task generator, a solver, and checks for whether generated goals can be reached. Task validation uses the generated rules, while the solver obtains information about their effects through public images and feedback. Tasks are grouped into multi-level sequences, called courses, that share game mechanics while changing their starting states, goals, or constraints. Figure 1 shows how an observed furnace cost guides production in a later task.

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.

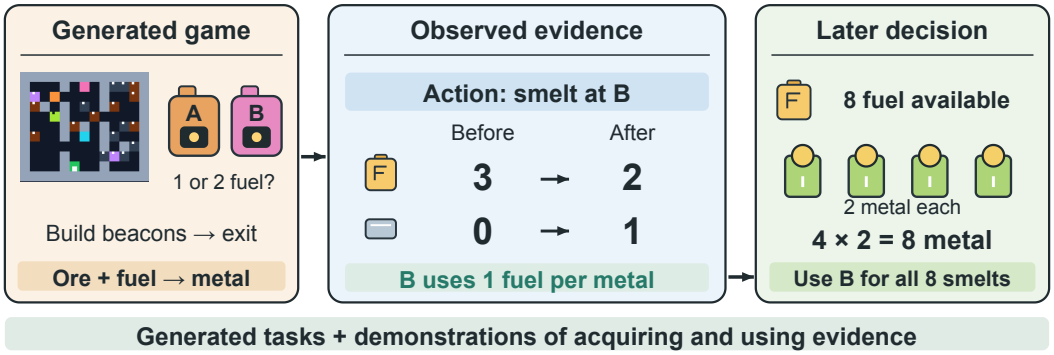


Figure 1: RISE generates tasks together with demonstrations of acquiring and using evidence. Left: the workshop converts ore and fuel into metal to build beacons before reaching the exit. Middle: smelting at furnace B reduces fuel from three to two and increases metal from zero to one. Right: the observed cost supports producing the eight metal needed for four beacons from eight fuel; this example uses a preconstructed resource course.

In five compositional game types, RISE uses observations from an early demonstration to construct later goals. The constructor selects a goal with a verified solution path that includes actions the solver has not yet tried in the corresponding states. The recorded observations support predictions of these outcomes and let the solver plan how to reach the later goal. The other four game types use fixed sequences linked by shared rules, resource limits, or movement patterns.

During collection, the solver tests actions whose effects remain uncertain and plans with the observations gathered so far. Replay checks reproduce the interactions, verify their outcomes, and trace structured evidence records to the observations that support them. Training targets include actions and updates to these records, so the data covers both gathering information and using it in later decisions. The solvers are reused across tasks and produce demonstrations without per-step LLM teacher calls.

We evaluate RISE on nine game types covering compositional puzzles, navigation, rule editing, resource dependencies, and tasks with timing constraints. Construction checks assess whether generated tasks and demonstrations execute correctly and whether exported training examples match the recorded interactions. A separate data-recipe study uses a sixteen-game implementation developed before the evidence-conditioned goal constructor. Synthetic supervision increases 8B level completion from 3.69% to 19.32%; a 4B recipe comparison increases completion from 9.66% to 17.33% at a similar target-token budget. The gains are concentrated in several game types and support the data recipe within this development setting. The studies use different game sets and interaction budgets, with their protocols and detailed outcomes reported separately.

Our contributions are:

- **Joint game and demonstration generation.** RISE pairs executable visual games with demonstrations collected through public images and feedback, using reusable task generators and solvers.
- **Using early observations to construct later tasks.** In five compositional game types, early observations guide the construction of later goals and support predictions for actions not yet tried in a given state.
- **Construction validation and data-recipe evaluation.** We assess generated data through execution checks and visual-agent experiments, including a recipe comparison with approximately matched target-token budgets that improves task completion on a sixteen-game development panel.

2 Related Work

Environment and task synthesis. Procgen studies generalization across generated levels, while PAIRED uses regret to guide environment construction (Cobbe et al., 2020; Dennis et al., 2020). AgentGen synthesizes environments and tasks, then uses FastDownward to generate planning demonstrations (Hu et al., 2025b). SPADE trains an environment designer through self-play; Echoverse uses graded interactions to improve executable environments, tasks, and agents (Liu et al., 2026; Pandya et al., 2026). Alchemy studies inference over episode-specific latent structure, while FALSIFYBENCH evaluates inductive rule discovery (Wang et al., 2021; Bertolazzi et al., 2026). RISE studies how public observations can support demonstration decisions and constrain the construction of later tasks. This couples the information exposed in a demonstration to the goals constructed from it.

Synthetic supervision and teacher information. STaR uses generated rationales and answer-based filtering, while AgentInstruct constructs instruction data through generative teaching workflows (Zelikman et al., 2022; Mitra et al., 2024). Learn-by-interact and AgentTrek obtain agent training data through environmental interaction and tutorial-guided replay, respectively (Su et al., 2025; Xu et al., 2025). DAgger and related work show why the states visited during collection matter for imitation learning (Ross et al., 2011; Ross & Bagnell, 2014). Work on privileged teachers addresses the gap between the information available during teaching and deployment (Chen et al., 2020; Warrington et al., 2021). RISE separates task verification using the true dynamics from demonstration generation using public images, feedback, and retained notes. Exact image decoding and bounded symbolic search are solver capabilities whose transfer to the learner is evaluated separately.

Feedback and long-horizon evaluation. ReAct interleaves reasoning and actions, and Reflexion uses feedback to form verbal memory for later attempts (Yao et al., 2023; Shinn et al., 2023). VAGEN studies world-model reasoning in multi-turn visual agents, while Tycho uses programmatic world models for active abstraction (Wang et al., 2026; Lehmann et al., 2026). AgentOdyssey generates long-horizon text games for continual learning (Zhang et al., 2026b). RISE produces training interactions containing observed effects, revisions, and retained evidence under a common visual interface. Shortcut learning and evaluation-protocol effects motivate examining how a task is solved alongside its final score (Geirhos et al., 2020; Machado et al., 2018). Autonomous play measures task completion, while paired-history diagnostics assess decisions conditioned on solver-generated evidence. BALROG and Crafter provide complementary contexts for testing interactive behavior beyond the synthesized environments (Paglieri et al., 2025; Hafner, 2021).

3 Method

RISE co-synthesizes executable environments, observation-grounded demonstrations, and linked task specifications. Construction has two linked requirements: tasks must be solvable under the generated dynamics, and demonstrations must support their choices with public observations. Figure 2 shows how observed evidence feeds back into task construction before verified interactions become training targets.

3.1 Environments and public interaction

An environment module κ implements a game type through states $s \in \mathcal{S}_\kappa$, actions $a \in \mathcal{A}_\kappa$, a renderer R_κ, and a parameterized transition law $T_{\kappa,\theta}$. The hidden mechanism θ determines an action's effects, and each level supplies an initial state, a goal, and interaction budgets. A course contains one or more levels governed by θ, with states or goals changing between levels.

The agent sees rendered images and public information about controls, goals, outcomes, and remaining budgets. The interaction interface projects public history h_t and retained agent-authored memory m_t into the next input:

$$z_t = \Pi_\kappa(h_t, m_t; b_{\kappa,\ell(t)}), \qquad u_t \sim \pi_\phi(\cdot \mid z_t), \tag{1}$$

where $\ell(t)$ is the current level, $b_{\kappa,\ell(t)}$ specifies its limits together with shared history, image, and memory allowances, and u_t is a tool command. Compared learners use the same observation fields

and view limits; collection and evaluation action budgets are specified separately. Neither a private state serialization nor the true mechanism is supplied to the learner. New and revised notes are retained automatically up to the memory capacity and can persist across context truncation, resets, and levels.

A replay-verified public-feedback solver calibrates action limits of 32 to 200 per level, including its exploration and recovery costs. For per-level-budget collection and evaluation, these limits are frozen before execution; resets consume budget without replenishing it (Appendix C).

3.2 Joint construction of mechanisms and tasks

A generator samples a mechanism and constructs initial states and goals for that mechanism. We instantiate this design on nine game types drawn from the wider library in Appendix F. Each type supplies executable semantics, rendering, task construction, and an accompanying solver. Five types use compositional transformations: sliding tiles, liquid measurement, disc transfer, cellular updates, and stack rearrangement. Four others add navigation, editable rules, resource dependencies, and temporal constraints. These nine types share interaction, validation, and export interfaces, with game-specific hypothesis representations and planning procedures.

For compositional transformations, each control is assigned a hidden typed program built from primitives p and state features f:

$$P ::= p \mid \mathrm{seq}(P, P) \mid \mathrm{when}(f, v, P) \mid \mathrm{repeat}(n, P). \tag{2}$$

Sequencing combines effects, conditions restrict their execution, and bounded repetition changes how many times an operation occurs. Numbered controls conceal these compositions, and optional failure conditions associate an action with a state feature. For these finite domains, a separate interpreter checks every compiled transition, and full transition tables identify equivalent behavior up to control permutation. Behavioral grouping prevents syntax changes and control renaming from being counted as new mechanisms.

The remaining four modules compute transitions on demand. Room mechanisms specify a key–door mapping and whether using a key consumes it. Word-rule mechanisms specify whether adjacent noun–property statements are read horizontally or along both axes. Moving words changes the active properties under this fixed reading convention. Resource mechanisms vary harvesting yield, tool wear, furnace efficiency, and processing costs in recipe graphs; courier mechanisms vary vehicle direction, speed, and collision along swept cells. These modules check executed transitions against independent implementations and replay feasible solution witnesses. They use structured mechanism identifiers and layout hashes rather than complete-domain equivalence classes.

3.3 Linking demonstrations to later tasks

A solution witness establishes that a goal can be reached under the true mechanism. A demonstration of information acquisition also needs actions whose outcomes distinguish plausible mechanisms. For a candidate pair, the generator seeks an action prefix executable from the same visible state under both mechanisms, followed by an action whose observed outcomes differ. Such a certificate identifies an available experiment and its contrasting outcomes. It does not imply that every successful policy must perform that experiment.

For compositional tasks, an early demonstration supplies the evidence used to construct a later goal. The constructor filters initial programs by this evidence, expands an empty set (Section 3.4), and selects a new start–goal pair. An accepted route includes an unexecuted transition whose outcome the evidence-consistent expanded candidates agree on, but the unconditioned initial catalog does not. Private transition checks verify the route, its length, and recoverability constraints. Replay verifies that supporting records remain available at the later level.

The four other modules pair levels through predefined dependencies while retaining the same hidden mechanism. A room course changes its layout or objective while preserving key semantics; a word-rule course changes the editing problem while preserving the reading convention. These tasks are constructed before demonstration collection and retain the mechanism across levels. Resource courses expose station costs with a generous first-level fuel budget, then require efficient production

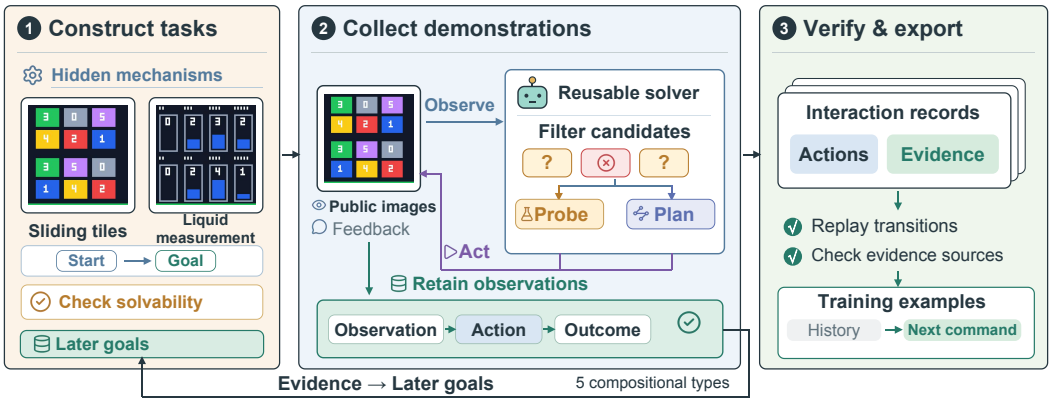


Figure 2: RISE constructs tasks, collects demonstrations, and verifies their export into training examples. The left panel checks task solvability under generated mechanisms; the middle panel shows a reusable solver filtering candidates, probing, and planning through public images and feedback. In five compositional types, the bottom return path uses retained observations to construct later goals. The right panel replays transitions and checks evidence sources before exporting history and next-command targets.

under a tighter later budget (Figure 1). Courier courses preserve motion and collision rules across changed routes and collection requirements. For word-rule tasks, exhaustive reachability with word movement disabled can additionally establish that editing is necessary. This property is distinct from needing to infer the hidden reading convention. Random and repeated-action policies (blind policies) provide auxiliary shortcut diagnostics under the same declared budgets. Their scores describe those policies, without certifying that every successful strategy uses evidence.

3.4 Demonstrations from public observations

Each game type supplies a solver with exact symbol decoding, public operations, and a bounded hypothesis space. Its episode-specific knowledge comes from accessible images, feedback, and retained observations. This procedure is reused across levels with interaction-dependent evidence and plans. Its exact decoder and symbolic search are not supplied to the trained visual agent. For a compositional control a, observed non-loss transitions $E_{a,t}$ constrain the candidate programs:

$$\mathcal{C}_{a,t} = \{P \in \Gamma_{a,t} : T_P(s,a) = s' \text{ for all } (s,a,s') \in E_{a,t}\}. \tag{3}$$

Here $\Gamma_{a,t}$ is the available program pool, initially a module-specific catalog. Loss observations update separate hazard candidates rather than being treated as ordinary state transitions. The other solvers maintain candidates for their module's latent parameters, using decoded states and public terminal feedback. A written explanation does not add an observation to the evidence set.

Planning and probing. The compositional solver seeks a goal route through observed safe transitions and agreed predictions. Prediction requires nonempty candidate sets that agree on the transition and its safety. Predictions are conditional on the candidate set containing the true mechanism. If no such goal route exists, the solver seeks a reachable state where probing separates candidates. Probe scores divide outcome entropy by travel, probing, and possible reset costs. After each action, the solver updates its knowledge from the resulting observation.

Search after counterexamples. When non-loss observations eliminate all initial programs, bounded composition search adds evidence-consistent candidates. It combines catalog programs with short primitive sequences; Appendix B specifies the bounds. Candidate consistency uses only observed non-loss transitions; the search receives no true program, private witness, or instance seed. Without candidates, it follows observed edges or probes untested transitions within budget; otherwise it stops. Empty search leaves the mechanism unresolved, without establishing task unsolvability.

Family-specific control and memory. The remaining solvers plan with candidates for key semantics, word interpretation, resource costs, or vehicle motion. Across modules, notes cite actual observations and retain useful effects for later decisions. After a loss, the solver can record the failure, reset within budget, and replan using retained evidence. Observed facts and candidate explanations have distinct labels and validation rules.

3.5 Verification and training supervision

The interface exposes six commands: `act`, `hypothesize`, `revise`, `remember`, `reset`, and `submit`. Ordinary actions need neither a rationale nor repeated memory identifiers; evidence updates cite their source observations. Optional rationales explain selected probes, revisions, or recovery decisions without becoming a separate hidden-state channel.

Auditing replays each course to reconstruct its images, tool feedback, outcomes, and projected prefixes. Structured transition facts must match executed events accessible in the current prefix or memory. Search records identify source observations; transfer checks verify evidence availability at course entry. Independent replay checks transitions; policy recomputation checks the recorded solver decisions. These checks verify structured records, not every sentence of a free-form explanation.

Replay-verified solver decisions from accepted demonstrations and corrective continuations become training targets. Corrective trajectories execute an erroneous or perturbed prefix before resuming the solver from that public history. Erroneous prefix commands are masked from the loss; subsequent solver decisions are supervised. Course and trajectory identifiers link each decision to its deduplicated target window. Only the current command u_i contributes to the training loss:

$$\mathcal{L}(\phi) = -\frac{1}{\sum_{i\in\mathcal{D}} |u_i|} \sum_{i\in\mathcal{D}} \sum_{j=1}^{|u_i|} \log p_\phi(u_{i,j} \mid z_i, u_{i,<j}). \tag{4}$$

Export and inference must produce matching input tokens and image tensors through the model's multimodal processor. Earlier commands and feedback remain context with their loss masked. The ledger distinguishes source courses, corrective variants, unique targets, and repeated target exposure. Appendix B specifies the solver bounds and the exported evidence fields.

Figure 3 shows the generated artifacts: rendered environments and an exported interaction target. The recipe example records a station cost from inventory changes and cites both source observations.

4 Experiments

We evaluate the coverage and reliability of generated environments and demonstrations, then examine how synthetic data recipes affect visual-agent execution. The construction audit uses nine game types, while the data-recipe study uses a separate sixteen-game implementation developed before evidence-conditioned follow-up goals. The latter tests synthetic supervision and recipe design; additional nine-type and four-type studies assess other supervision settings in Appendix C.1.

4.1 Generated resources and coverage

The library contains 40 implementations grouped into 32 gameplay families, with nine types equipped for the construction and supervision pipeline studied here. The nine-type corpus contains 183 source courses, 230 recorded episodes including corrective continuations, and 13,928 distinct supervised windows (Figure 4). Deduplication compares model-visible inputs, images, and target commands; batch alignment adds 24 repeated windows to the training exposure.

The source-course count is nearly balanced across game types, with 20 or 21 courses each, while the number of supervised windows varies with the collected interactions. Actions account for 9,396 windows, revisions for 3,615, hypotheses for 798, and resets for 119. Hypothesis and revision targets record information gathered during interaction, giving the corpus supervision beyond the next physical action.

The corpus has 214 distinct rendered initial frames and 101 exhaustive transition-table classes across the five finite-state types. The transition-class count remains 101 after merging action-label

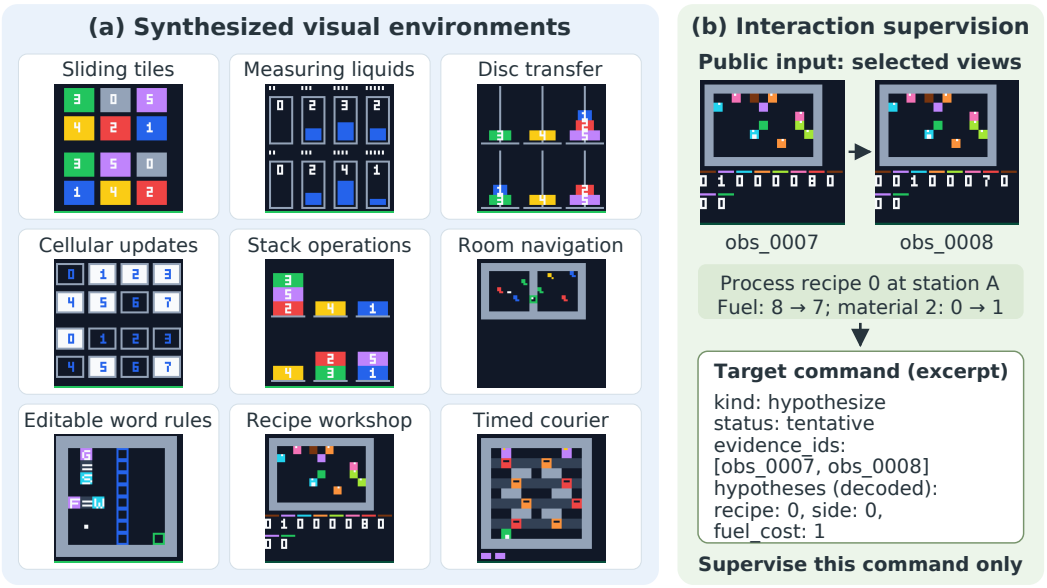


Figure 3: Generated environments and supervision from the experimental subset. Left: public renderings from the nine game types, including a recipe-graph workshop. Right: two retained views from an interaction prefix and selected fields of its next command. The fuel and inventory changes support a station-cost record; only the current command contributes to the loss. Hypothesis fields are decoded for display.

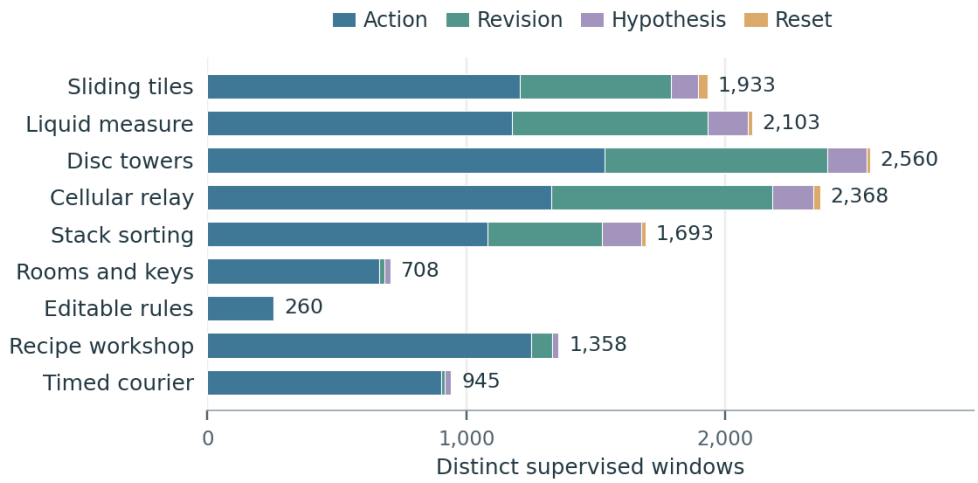


Figure 4: Composition of the nine-type training corpus: each bar counts distinct supervised windows by target-command type, with totals shown at the right.

permutations; the other four types report parameter signatures instead (Appendix B.1). Among 180 clean solver episodes, full-course length has a median of 58 tool decisions and a 95th percentile of 121, including evidence updates and environment actions.

4.2 Construction reliability

A fixed-pool audit attempts 100 candidates per game type, giving 900 attempts in total. The generator admits 507 candidates, all of which pass transition, image, and public-prefix replay checks; 501

complete interaction and export within 400 tool decisions. Complete-export yield ranges from 7/100 for sliding tiles to 100/100 for each of three fixed-course types. Appendix B.1 reports the rejection categories and a separate per-level-budget validation, which accepts 29 of 36 assigned episodes on existing source courses.

Controlled fault tests detect all 60 injected transition, rendering, probe-partition, and citation-access errors, while accepting all 40 valid controls. These tests measure detection of the specified faults; valid citations alone do not establish the truth of arbitrary explanatory text (Appendix B.2).

4.3 Learning from synthesized data

The data-recipe study compares Qwen3-VL-Instruct models before and after supervised fine-tuning on a sixteen-game generator (Bai et al., 2025). For 4B, the standard and revised recipes use the same game coverage and per-game target-token quotas. The revised recipe reallocates supervision toward exploration, recovery, continuation after failed attempts, and memory-related decisions while retaining successful execution examples. Erroneous actions appear only in the input history; training loss applies to the final target command. The recipes contain 995,835 and 998,741 target tokens, respectively, differing by 0.29%; trajectories, input exposure, and optimizer updates also differ.

All five conditions face the same 169 visual tasks from 16 game types, comprising 68 single-level tasks, 60 two-level courses, and 41 four-level courses. There are 352 planned level positions; later levels remain in the denominator when an earlier level is not completed. Each level allows 64 model calls and two resets, with identical initial inputs and interfaces within each comparison. Fifteen game types reuse development-diagnostic tasks, while Rule Workshop uses held-out layouts; this is a development study with one training seed per recipe.

Figure 5 shows how the 169 assigned tasks are distributed across games and course lengths. Twelve game types include four-level courses; balance dispatch and clock circuit use one- and two-level tasks, chromatic flood uses one- and two-level tasks, and factor sequence uses single-level tasks. Multi-level courses contribute 284 of the 352 planned positions, or 80.7%, so overall level completion includes substantial weight from connected tasks.

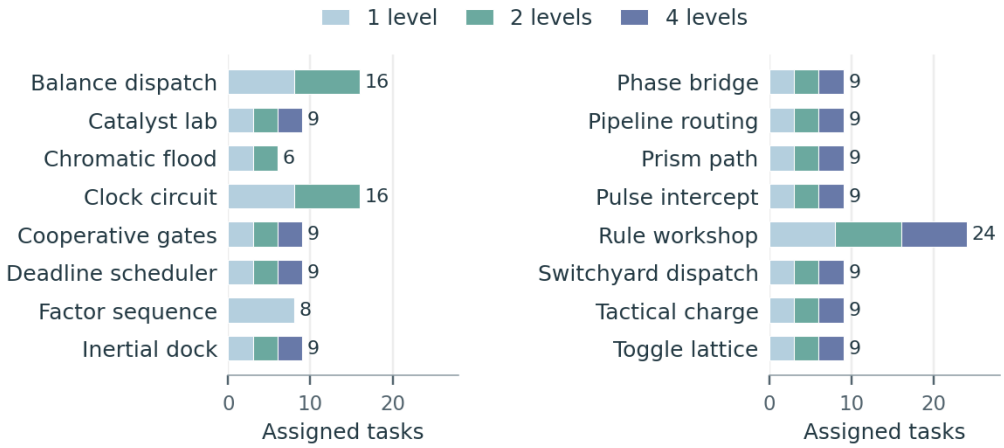


Figure 5: Task coverage in the sixteen-game development study: colored segments count courses with one, two, or four levels; all five model conditions use the same 169 tasks.

Synthetic training increases 8B level completion from 13/352 to 68/352, a gain of 15.63 percentage points (Table 1). For 4B, revising the recipe increases completion from 34/352 to 61/352, or 7.67 percentage points. Both trained 4B models have zero rejected commands, so the difference extends beyond whether commands satisfy the interface format. The revised 4B model improves course progress on 22 tasks and regresses on five; Deadline Scheduler, Catalyst Lab, and Rule Workshop account for 25 of the 27 additional completed levels. Table 3 reports costs and paired outcomes; the supplement provides the complete per-game results.

Table 1: Data-recipe study on 169 tasks and 352 planned level positions from a separate sixteen-game implementation; Instruct denotes the unmodified instruction-tuned model; game macro averages course progress within each game and then across games.

Model	Data recipe	Levels	Completion (%)	Game macro (%)	Tasks
4B	Instruct	18 / 352	5.11	6.77	5 / 169
4B	Standard SFT	34 / 352	9.66	11.85	14 / 169
4B	Revised SFT	61 / 352	17.33	19.44	29 / 169
8B	Instruct	13 / 352	3.69	4.34	3 / 169
8B	Revised SFT	68 / 352	19.32	21.57	28 / 169

Table 2 separates the same assigned tasks by their number of levels. The revised 4B recipe completes more levels in each group: 17 versus 11 in single-level tasks, 26 versus 12 in two-level courses, and 18 versus 11 in four-level courses. The trained 8B model completes 25 of the 164 planned positions in four-level courses, compared with seven before training. These groups contain different tasks, so the breakdown describes where gains occur within this suite.

Table 2: Completed levels by task length in the sixteen-game study; each row uses its shared planned-level denominator and includes unreached later levels.

Levels/task	Planned	4B Instruct	4B standard	4B revised	8B Instruct	8B revised
1	68	5	11	17	3	18
2	120	4	12	26	3	25
4	164	9	11	18	7	25

The interaction ledger records model calls, physical actions, resets, rejected commands, and input/output tokens for all 169 assigned tasks. These quantities separate execution quality from the cost of producing each trajectory.

Table 3: Total interaction costs over all 169 assigned tasks per condition, including failures.

Model / recipe	Calls	Actions	Resets	Rejected	Input tokens	Output tokens
4B / Instruct	6,554	2,393	201	3,801	26,990,739	693,233
4B / Standard SFT	4,477	4,171	306	0	35,177,102	466,996
4B / Revised SFT	4,399	4,122	277	0	34,737,230	475,097
8B / Instruct	4,159	3,688	259	168	27,153,609	166,519
8B / Revised SFT	4,358	4,074	281	3	34,298,802	476,353

4.4 Learning across supervision settings

The sixteen-game study demonstrates an execution benefit from synthetic supervision and from revising the overall data recipe. During evaluation, the trained models solve tasks through action and reset commands. Complementary nine-type and four-type comparisons find low task completion and no consistent full-supervision advantage (Appendix C.1). Together, the studies establish executable, traceable data construction and show that data allocation affects learning outcomes in the sixteen-game development setting.

5 Conclusion

RISE co-synthesizes executable environments, demonstrations, and linked tasks for long-horizon visual interaction. Its technical focus is the connection between publicly observed evidence, demonstration decisions, and later-task construction. Construction audits establish executable tasks and traceable structured records; a separate sixteen-game development study shows improved execution after synthetic training and further gains from data-recipe revision.

AI USE STATEMENT

We used AI tools for literature discovery, research development, and manuscript drafting and polishing.

REFERENCES

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

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

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Leonardo Bertolazzi, Katya Tentori, and Raffaella Bernardi. Falsifybench: Evaluating inductive reasoning in llms with rule discovery games. *arXiv preprint arXiv:2606.04751*, 2026.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on robot learning*, pp. 66–75. PMLR, 2020.

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

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

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

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in neural information processing systems*, 36:73383–73394, 2023.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

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

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

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

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

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

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

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

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 496–507, 2025b.

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

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

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, et al. Spade: Self-play in adaptive synthetic executable environments. *arXiv preprint arXiv:2608.19197*, 2026.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

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

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

Yash Pandya, Sahil Gupta, Sarthak Harne, Archana Yadav, Kavyansh Chourasia, Hussein Mozannar, Vibhav Vineet, Sara Abdali, Corby Rosset, Yash Lara, et al. Echoverse: Deep, evolving environments for training computer-use agents at scale. *arXiv preprint arXiv:2607.28074*, 2026.

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

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

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *Advances in Neural Information Processing Systems*, 38, 2026.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Arik. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *International Conference on Learning Representations*, volume 2025, pp. 89044–89097, 2025.

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

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 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*, 2024.

Jane X Wang, Michael King, Nicolas Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, Francis Song, et al. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents. *arXiv preprint arXiv:2102.02926*, 2021.

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

Andrew Warrington, Jonathan W Lavington, Adam Scibior, Mark Schmidt, and Frank Wood. Robust asymmetric learning in pomdps. In *International Conference on Machine Learning*, pp. 11013–11023. PMLR, 2021.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, et al. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*, 2025.

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

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025.

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

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

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

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Kangning Zhang, Shuai Shao, Qingyao Li, Jianghao Lin, Lingyue Fu, Shijian Wang, Wenxiang Jiao, Yuan Lu, Weiwen Liu, Weinan Zhang, et al. Mmskills: Towards multimodal skills for general visual agents. *arXiv preprint arXiv:2605.13527*, 2026a.

Zheyuan Zhang, Zehao Wen, Alvin Zhang, Andrew Wang, Jianwen Xie, Daniel Khashabi, and Tianmin Shu. Agentodyssey: Open-ended long-horizon text game generation for test-time continual learning agents. *arXiv preprint arXiv:2606.24893*, 2026b.

A Environment modules and validation

Table 4: Experimental environments and their roles in rule inference and evidence reuse. The five compositional modules share one row; each other module has its own.

Environment	Hidden mechanism	Public evidence	Later-task link
Transformations	Composed control programs	State changes; loss	Evidence-conditioned goals
Room navigation	Key mapping; consumption	Key use and door outcomes	Shared key semantics
Editable word rules	Reading convention	Edited rules and outcomes	New editing objectives
Resource workshop	Yield; wear; processing cost	Inventory changes	Tight fuel constraints
Timed courier	Direction; speed; collision	Motion and collision outcomes	New collection routes

Table 5 summarizes the nine experimental game types and their hidden mechanisms. The five finite domains count ordinary states before an additional absorbing failure layer. Their generation grammar has maximum depth three, with the root at depth zero, and repetition counts of two or three. Conditions compare a state feature's parity with zero or one. The compiler and an independently implemented stack interpreter must agree on every state and control. Identity controls, duplicate controls within a mechanism, and mechanisms with more than 70% unchanged state–control transitions are rejected.

Table 5: Experimental environment modules. State counts apply to the five finite transformation domains; the other modules generate layouts and compute transitions on demand. Resource variants expose seven or eight actions.

Structure	States	Actions	Hidden mechanism
Sliding tiles	720	4	Composed primitive programs
Liquid measurement	52	6	Composed primitive programs
Disc transfer	243	6	Composed primitive programs
Cellular updates	256	6	Composed primitive programs
Stack rearrangement	2,160	6	Composed primitive programs
Room navigation	Not enumerated	6	Key mapping and consumption
Editable word rules	Not enumerated	4	Horizontal or two-axis reading
Resource workshop	Not enumerated	7–8	Yield, wear, processing costs
Timed courier	Not enumerated	5	Direction, speed, swept collision

Finite-mechanism signatures sort complete control transition tables before hashing them with the structure identifier. The base signature is assigned before hazard variants, keeping related tasks in the same split. Safe breadth-first search supplies shortest witnesses for these finite domains. Base task construction requires at least 24 safe reachable states and goal recoverability from at least 90% of them. Evidence-conditioned transfer requires observations that eliminate every initial program for a control, followed by search that finds consistent candidates. At an unexecuted state–action edge, the expanded candidates must agree where the initial catalog, before conditioning on the collected evidence, admits different outcomes. The later goal has a longer shortest safe path than the teaching goal; its route planned from retained evidence is checked against the true transitions.

The room module uses six actions: turn left, turn right, move forward, pick up, drop, and open. Its state contains position, heading, a single carried item, and the map; the public renderer shows the full map. Three key types admit six bijective door mappings, combined with two consumption settings for twelve labeled mechanism candidates. Task constructors vary room topology and use fetch, return, put-next, and exit objectives. Feasible paths are replayed with independent transition semantics; their lengths are witness lengths without a global shortest-path claim.

The word-rule module uses four movement actions and three nouns: gate, block, and flag. Adjacent noun–property statements assign STOP, PUSH, WIN, or HURT, with properties recomputed after movement. Push chains move as a whole or remain stationary, HURT takes precedence over WIN, and marked words are fixed. The hidden reading convention has two candidates: horizontal-only

and both-axes. In editable-rule games, word positions and the properties they induce belong to s; the hidden convention for interpreting those words remains fixed within the course. Six task recipes combine rule removal, rule construction, pushing, hazards, and compound dependencies. Most layouts use randomized tree-based terrain with added cycles; the compound recipe uses a dedicated two-barrier template. Editing-necessity checks disable word movement and exhaust the resulting reachable graph before concluding that no winning state exists.

The beacon workshop variant uses four movement actions plus harvesting, tool crafting, smelting, and beacon construction. Typed inventory transactions check input quantities, output capacity, and empty-tool preconditions before applying any change. The hidden axes are wood yield, tool wear, and which furnace consumes one rather than two fuel per metal. An independent implementation checks transitions by direct inventory accounting.

For the tight-budget resource level, let F be available fuel, M current metal, and R the metal required by unlit beacons. Four beacons require two metal each, and the initial state has $F = 8$, $M = 0$, and $R = 8$. Efficient smelting and beacon construction preserve $F + M - R$, whereas an inefficient smelt decreases it by one. Within an attempt, no other action increases this quantity, so a negative value certifies an irrecoverable resource shortfall and triggers loss. This establishes the need for efficient smelting in a successful attempt; it does not require every successful policy to infer the furnace identity, since a fixed choice may be correct.

The recipe-graph variant composes chains, branches, and shared intermediates into acyclic production graphs with one producer per material. Its seven actions comprise movement, harvesting, processing, and delivery; the public manual specifies the graph, while each recipe independently assigns costs of one and two fuel to its two stations. Reverse expansion of the target, after subtracting available inventory, yields the exact remaining number of processing operations C under this grammar. A teaching level supplies $2C$ fuel and a later level supplies C, so an inefficient operation in the latter reduces fuel below the remaining requirement. Canonical graph fingerprints preserve input identity and shared dependencies when removing topology aliases.

The timed courier uses four movement actions and waiting; every action advances the vehicles. The courier moves first, after which vehicles move with a shared speed and alternating lane directions. Contact with a current or final vehicle position causes loss; whether intermediate swept cells also cause loss is hidden. The reference implementation advances vehicles in microsteps to check the production transition calculation. Collision-setting aliases at speed one are merged before collection, leaving six distinct motion settings. Resource and courier courses use predefined level dependencies, with feasible witnesses checked by independent replay.

B SOLVER POLICY AND SUPERVISION

The compositional solver builds its initial catalog independently of the episode seed and true program. It enumerates primitives, ordered primitive pairs, feature-conditioned primitives, and repeated primitives, then applies additional bounded wrapping and composition. Small primitive libraries also admit enumerated sequences of three to five primitives. Full transition tables remove behaviorally duplicate candidates, including identity behavior. The environment grammar can generate a control absent from this initial catalog.

Counterexample-triggered search appends primitive sequences of length at most three to initial-catalog programs. It visits at most 4,096 behaviorally distinct suffix nodes and returns at most 256 distinct candidate programs. Complete transition tables support deduplication, while consistency uses only the supplied non-loss transitions. The report records the evidence hash, node count, candidate count, and stop reason. Exhaustion covers only the declared bounded search space and does not establish rule impossibility. The search does not introduce new primitive semantics or expand the hazard hypothesis space. Probe selection uses finite-candidate outcome entropy as a heuristic, without a calibrated posterior. The word-rule solver seeks a distinguishing probe or an editing prefix safe under all surviving conventions until the next informative outcome. The resource solver updates recipe costs from inventory changes; the courier solver observes motion from safe locations and plans over time-indexed states, with budgeted collision tests when needed.

A stack example illustrates why the location of a probe matters. Consider a primitive that moves the top object from one stack to another, and a candidate that executes it twice. If the source contains

one object, the second move is a no-op, so both candidates produce the same observation. With two source objects, their outcomes differ. The solver therefore needs a route to a suitable test state, and the cost of that route affects which experiment is useful. A retained observation can then rule out a candidate when the control is used in another configuration.

Table 6: Public-evidence decision procedure for compositional environments. Each environment action is followed by a new observation before the procedure repeats. The other four solvers use their own candidate representations and planners.

Step	Decision from public observations and retained records
1	Record new non-loss transitions or loss events; retain their source observations.
2	Update program and hazard candidates separately. Expand programs only after non-loss transitions refute the initial catalog.
3	Handle a terminal state using the available reset or submission command.
4	Seek a route on observed safe edges or predictions from nonempty program and hazard sets, with unanimous transitions and safety under every hazard candidate.
5	Otherwise, seek a distinguishing probe and include the cost of reaching it, acting, and resetting if needed.
6	If no candidate remains, retain observed edges and test an unobserved state–action pair if action and reset budgets allow; otherwise stop.
7	Execute one action, read the actual outcome, and repeat.

In compositional supervision, a supported record stores executed transition facts rather than a representative program. Candidate explanations can appear in short rationales and must remain consistent with their cited examples. Word-rule notes store a candidate reading convention with its actual source observations and retain a tentative label. Audits check source identifiers, accessible evidence, executed outcomes, and the declared policy's recomputed choices. For the other four game types, policy checks rerun the same solver implementation, while environment transition replay uses independent semantics.

New and revised notes enter bounded memory automatically; exceeding capacity evicts the oldest retained note. The optional `remember` command changes the retained selection, while storage leaves a note's truth status unchanged. The program solver writes concise observation records using a reversible, game-specific fact grammar. This conversion preserves the recorded facts and adds no hidden state or future outcome. An otherwise valid action with an unavailable optional memory reference executes unchanged and receives an attribution warning. Invalid actions, malformed commands, and unavailable observation citations remain subject to rejection. The history preserves the original command and feedback, so a warning cannot silently replace the learner's output with a corrected answer. Corrective data replay the erroneous prefix, recompute the solver continuation, and verify the resulting physical trajectory and public inputs.

The training ledger records source courses, mechanism groups, full trajectories, unique targets, and target-token totals. The frozen corpus uses exact input, image, and target deduplication, with source-course components kept together when partitioning records. A command family with no executed examples is recorded as uncovered rather than inferred from the interface's availability. Coverage reports distinguish executed losses and resets from protocol rejections and repeated ineffective actions. Repeated prefixes share a target only when their model-visible inputs and commands agree; trajectory indices retain every occurrence. Native rows contain public messages, image references, and tool schemas, with only the current command unmasked for loss. Export admits only complete commands that satisfy the configured input and target budgets; over-budget rows are rejected.

B.1 Corpus and construction checks

Table 8 breaks down the frozen corpus by game type before repeated exposure. The 13,928 distinct windows contain 9,396 action targets, 3,615 revisions, 798 hypotheses, and 119 resets. Remember and submit are available commands but have no targets in this corpus. All exported windows passed input-token and image equality checks and applied loss only to the current command. Table 9 reports 214 distinct rendered initial frames and 101 exhaustive transition classes in the five finite-state types. The class count remains 101 after identifying control permutations; each table includes the absorbing failure layer. The four spatial types report configuration signatures: key semantics, reading convention, recipe graph with processing rules, or motion rules with collection order. These signatures record

Table 7: Nine-type corpus accounting, separating distinct records from repeated training exposure.

Unit	Training corpus
Game types	9
Source courses	183
Recorded episodes	230
Clean episodes	180
Corrective episodes	50
Distinct supervised windows	13,928
Distinct target tokens	1,208,326
Window exposures	13,952
Exposed target tokens	1,210,494

specified configuration fields; only the finite-state counts establish exhaustive transition equivalence. Initial-frame counts use native rendered pixels, including visible goals, so visually identical starts can still carry different textual missions.

Table 8: Frozen training corpus by game type. Window and target-token counts are deduplicated and exclude batch-alignment repetitions.

Game type	Courses	Episodes	Windows	Target tokens
Sliding tiles	20	26	1,933	190,351
Liquid measure	20	26	2,103	200,745
Disc towers	20	26	2,560	304,582
Cellular relay	21	27	2,368	217,807
Stack sorting	20	26	1,693	163,532
Rooms and keys	21	25	708	27,163
Editable rules	21	26	260	10,513
Recipe workshop	20	24	1,358	59,931
Timed courier	20	24	945	33,702
Total	183	230	13,928	1,208,326

Table 9: Structural, behavioral, and visual coverage of the frozen training corpus. Transition classes apply to finite-state types; configuration signatures for spatial types include the specified rule and task fields.

Game type	Initial frames	Transition classes	Config. signatures
Sliding tiles	30	20	n/a
Liquid measure	30	20	n/a
Disc towers	28	20	n/a
Cellular relay	29	21	n/a
Stack sorting	28	20	n/a
Rooms and keys	11	n/a	9
Editable rules	22	n/a	2
Recipe workshop	24	n/a	20
Timed courier	12	n/a	11

The construction audit uses 100 attempted candidates per game type and a 400-decision solver cap. It rejected 202 candidates with duplicate controls, 83 with insufficient teaching goals, 76 outside the declared teaching prior, and 32 with identity controls. The remaining 507 passed syntax, independent transition and image checks, and public-prefix replay; six failed to complete within the decision cap. Table 10 reports the resulting 501 complete exports by type. These counts describe the construction pipeline with its admission rules active, rather than an ablation of individual checks.

Table 11 evaluates per-level-budget export on two existing source courses per game type. One sliding-tile course failed calibration, excluding its two assigned episodes; four recovery attempts lacked the required observed obstruction, and one did not complete. The 17 calibrated courses contain 31 reference levels, with limits of 32, 48, 64, or 96 actions in this validation. Recovery acceptance

Table 10: Generation audit with fixed attempt counts. Admitted candidates pass the generator filters; exported candidates additionally complete solver interaction and native target export within 400 decisions.

Game type	Attempted	Admitted	Exported
Sliding tiles	100	7	7
Liquid measure	100	32	29
Disc towers	100	21	21
Cellular relay	100	23	22
Stack sorting	100	24	23
Rooms and keys	100	100	99
Editable rules	100	100	100
Recipe workshop	100	100	100
Timed courier	100	100	100
Total	900	507	501

requires an executed obstruction, a changed continuation, and successful replay within the assigned budgets.

Table 11: Per-level-budget validation on 18 existing training-source courses. Each course has one assigned clean and one assigned corrective episode, giving 36 planned episodes and 29 accepted episodes.

Game type	Assigned courses	Calibrated	Clean	Corrective
Sliding tiles	2	1	1	1
Liquid measure	2	2	2	2
Disc towers	2	2	2	2
Cellular relay	2	2	2	1
Stack sorting	2	2	2	2
Rooms and keys	2	2	2	1
Editable rules	2	2	2	1
Recipe workshop	2	2	2	2
Timed courier	2	2	2	0
Total	18	17	17	12

B.2 Controlled fault tests

Table 12 evaluates predefined transition, rendering, probe-partition, and citation-access faults against valid controls. The four checks detect all 60 injected faults and accept all 40 valid controls. A separate test supplies ten false free-text claims with valid source access; citation validation accepts all ten. Structured transition facts can be replayed against observations, whereas unrestricted explanations require a separate semantic check.

Table 12: Fault detection with valid controls and injected errors; the final row tests claim truth beyond citation access.

Check	Valid	Accepted	Injected	Detected
Transition consistency	10	10	30	30
Image round trip	10	10	10	10
Probe partition	10	10	10	10
Citation access	10	10	10	10
Free-text claim truth	0	0	10	0

C Evaluation protocols

For structure k, let $\mathcal{Q}_k$ contain assigned courses, with L_q planned levels and c_q completed levels. Macro progress is

$$\text{Progress} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\mathcal{Q}_k|}\sum_{q\in\mathcal{Q}_k}\frac{c_q}{L_q}. \tag{5}$$

Planned-level completion is $\sum_q c_q / \sum_q L_q$, and complete-course success is the fraction satisfying $c_q = L_q$. A level left unreached by model failure or stopping remains in the planned denominator with zero completion. Protocol rejections, environment failures, and infrastructure failures are recorded separately. Paired model comparisons require completed records for every assigned course in both conditions.

For the nine-type study, action caps are calibrated from a completed, independently replayed public-feedback solver trajectory in the same course context. For reference action count n, the cap is the smallest value in $\{32, 48, 64, 96, 128, 160, 200\}$ covering $n + \max(8, \lceil n/4 \rceil)$. The call cap is $\lceil 1.5b \rceil$ for action cap b, with at most three resets; the reference must satisfy all three limits. A task exceeding the available caps is rejected, redesigned, or split before the evaluation manifest is frozen. Executed actions and resets count even when the state is unchanged; resets restore the level state but do not refill the remaining interaction allowance. The environment step limit matches this action cap, and each new level receives its own frozen allowance. Models see the remaining budget, but not the reference trajectory, reference action count, or private solution certificate. The nine-type export and learner evaluation share this controller when per-level limits are enabled; the original corpus retains its course-level collection limits.

For level ℓ, let a_ℓ^{ref} and a_ℓ be the reference and evaluated action counts, including exploration and recovery. Success within all frozen limits receives $E_\ell = 100 \min\{1, (a_\ell^{\text{ref}}/a_\ell)^2\}$; other model outcomes receive zero. The squared ratio follows ARC-AGI-3's action-efficiency design, replacing its human reference with a public-feedback solver and using a cap of one and equal level weights. Scores are averaged over assigned levels, then courses, then game types; model failures and unreached levels score zero, while infrastructure failures remain unresolved. Raw actions on successful levels, tool calls, and token usage accompany this score; external benchmarks retain their native budgets and metrics.

The nine-type supervision comparison matches source courses, initialization, trainable components, and target-token exposure. Distinct targets, repeated windows, input tokens, images, optimizer updates, and runtime remain separate accounting fields. The frozen corpus keeps its original course-level budgets, while both learners use the same calibrated per-level evaluation budgets.

Action-only histories are regenerated by executing stripped action, reset, and submission commands in a separate session. Replay must preserve the physical trajectory, environment feedback, completion outcomes, and action and reset costs. A second replay verifies every regenerated input prefix and command response. Action-only targets are repeated within game types to match target-token exposure; repeated windows, optimizer steps, input tokens, and runtime are reported separately. Removing note commands also changes which physical actions fit within a fixed history window, so this comparison concerns the complete supervision format.

All evaluation assignments are fixed before comparing checkpoints. The nine-type development comparison contains 39 non-overlapping courses and 67 planned levels. With both models completing zero levels, these observations do not establish equivalence between the supervision methods.

Paired histories are constructed by swapping control identities while preserving the initial state and goal. Of 102 predetermined constructions, 26 pass the evidence and execution checks; retaining at most one per source course leaves 15 pairs: six sliding-tile, one disc-transfer, one cellular, and seven stack-sorting pairs. Each branch is executed by the public-observation solver, independently replayed, and reset to the same rendered current state before querying the learner. The scoring set contains every first action on a shortest route supported by the public evidence; the two branches must have disjoint sets. Each model receives the same 60 branch/rendering inputs and may emit up to four commands before its first environment action, which is scored. History lengths and contents may differ across branches; these tests condition on supplied solver histories, while autonomous evaluation accepts any legal strategy that solves the task. The neutral border preserves the original pixels, and both the original and perturbed action must be correct and identical to earn border credit. Full supervision yields zero jointly correct history pairs; its action is unchanged in 24 of 30 border pairs, with six both unchanged and correct.

The internal assignment contains non-overlapping development courses, training-source diagnostics, and source-overlapping development diagnostics; the BabyAI assignment contains ten episodes. Task membership and budgets are shared within each paired comparison, with overlapping courses excluded from the main development result. BabyAI keeps its native task-success outcome and adapter limits, without aggregation with internal progress or action efficiency. The ten BabyAI episodes form a development diagnostic; task-level uncertainty does not estimate variation across independent training seeds.

C.1 Additional supervision studies

The nine-type comparison uses the 183-course corpus and the two final 8B checkpoints in Table 17. Both conditions complete zero of 67 planned levels on the 39 non-overlapping development courses (Table 13). The action-only and full models complete 8/18 and 2/18 single-level training-source tasks, respectively; both complete 0/10 levels on five source-overlapping development courses. The different collection and evaluation budgets and the limited training coverage constrain interpretation of these failures.

Table 13: Nine-type development results: progress and efficiency are percentages, levels and courses give completed/assigned counts, and calls are the mean per assigned course.

Supervision	Progress	Levels	Courses	Efficiency	Calls
Action-only	0.0	0 / 67	0 / 39	0.0	63.2
Full	0.0	0 / 67	0 / 39	0.0	92.3

Table 14: Nine-type paired-history and neutral-border diagnostics in percent, based on 15 history pairs and 30 border pairs.

Diagnostic	Action-only	Full
Both history branches correct	13.3	0.0
Border: unchanged and correct	23.3	20.0

Table 15: Ten-episode BabyAI diagnostic for the original nine-type checkpoints, using the same visual adapter within the comparison.

Supervision	Successful tasks / assigned tasks	Success (%)
Action-only	4 / 10	40.0
Full	0 / 10	0.0

An independent four-type pilot uses 53 source courses and compares Qwen3-VL-8B under action-only, full, and compact-note supervision, each with 3,360 physical-action target exposures. The respective update counts are 105, 175, and 142; target-token budgets are not matched. On 24 courses with 36 planned levels, the conditions complete two, two, and zero levels, all successes occurring on training-source tasks (Table 16). The same checkpoints complete 2/50, 1/50, and 3/50 BabyAI episodes, and 0/12, 1/12, and 0/12 Baba episodes under the visual adapter. BabyAI uses conditioned MixedTrainLocal tasks with zero distractors and limits of 128 physical actions and 160 calls; Baba allows 100 actions and 128 calls, and both adapters disable resets and retain earlier native termination. These differences do not establish a consistent benefit of full or compact-note supervision.

Table 16: Independent four-type pilot and its external diagnostics; all entries are completed/assigned counts.

Supervision	Internal levels	Complete courses	BabyAI	Baba
Action-only	2 / 36	0 / 24	2 / 50	0 / 12
Full	2 / 36	0 / 24	1 / 50	1 / 12
Compact notes	0 / 36	0 / 24	3 / 50	0 / 12

D TRAINING CONFIGURATION AND SCOPE

Table 17 specifies the frozen corpus, training configuration, and interaction limits for the nine-type supervision comparison. Uniform loss is applied over the unmasked target tokens. Full and action-only supervision use their final checkpoints after 436 and 1,224 updates, respectively. The first fully published final checkpoint is selected before this evaluation, without selecting an intermediate checkpoint by task score. Completed optimizer updates and the released checkpoint are bound to each reported evaluation result. Target-token matching does not imply equal updates or compute because the command formats differ in length.

Table 17: Training and interaction configuration for the two supervision conditions. The frozen corpus retains course-level collection limits; learner evaluation uses independently calibrated per-level limits.

Setting	Value
Learner	Qwen3-VL-8B-Instruct
Corpus	Nine-game corpus in Table 7
Updated components	Language model and multimodal aligner; vision encoder frozen
Precision / parallelism	BF16 / ZeRO-2
Optimizer	AdamW (fused); $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$
Learning rate / weight decay	0.000001 / 0.01
Schedule / warmup	Cosine decay / first 3% of updates, rounded up
Effective batch size	32
Update budget: full / action-only	436 / 1,224
Target exposure: full / action-only	1,210,494 / 1,210,650
Context / output limit (tokens)	32,768 / 1,024
History / images / notes	16 / 4 / 24
Collection actions / calls / resets	2,048 / 3,072 / 8 per course
Evaluation actions / calls / resets	Calibrated per level; Appendix C
Training runtime	Swift 4.5.0; PyTorch 2.8.0+cu126; Transformers 4.57.3; DeepSpeed 0.18.2

Adding a game type to the training pipeline requires executable semantics, rendering, a public decoder, task construction, and a solver policy. The other four game types use finite candidate sets for key semantics, word interpretation, resource costs, and vehicle motion; their manuals specify the operations. The learning evaluation concerns visual model behavior; exact decoding and symbolic search remain capabilities of the demonstration solver.

D.1 SIXTEEN-GAME DATA-RECIPE STUDY

The sixteen-game generator produces observation-conditioned demonstrations through exact image decoding and programmatic planning over finite mechanism families; it predates the follow-up-goal constructor in Section 3. The standard and revised recipes contain 8,864 and 8,608 supervised windows and 995,835 and 998,741 target tokens, respectively. Training uses a learning rate of 10^{-5}, an effective batch size of 32, and one traversal of the selected windows, giving 277 standard-recipe updates and 269 revised-recipe updates. The language model and visual connector are updated while the vision encoder is frozen, with training seed 20260915. The comparison changes the overall data allocation rather than a single supervision component.

Evaluation uses screenshots and English interaction history, with temperature zero, a 32,768-token context, a 4,096-token output limit, four images, and 16 recent decisions. Each level allows at most 64 model calls and two resets; engine action limits range from four to 48 per attempt. A reset starts

another attempt under the same remaining call allowance, unlike the cumulative per-level action rule used for the nine-type evaluation. All five conditions share the same tasks and initial inputs; the unmodified and trained models receive the same visual interface. Fifteen game types reuse development-diagnostic tasks, while Rule Workshop uses held-out layouts; no separate confirmation set or repeated training seeds are used.

The revised 8B model improves course progress on 45 tasks and regresses on three, with 121 ties; the 4B recipe comparison has 22 improvements, five regressions, and 142 ties. The trained 8B model uses 476,353 output tokens compared with 166,519 for Instruct; Table 3 reports both completion-related interaction and token costs. The trained models issue action and reset commands but no accepted explicit hypothesis, revision, or memory commands during this evaluation. The observed gains concern the complete data recipe and its stated development tasks.

E Connections to agent training and evaluation

Learning through games and generated tasks. Game-based training provides controllable settings for learning perception, reasoning, and sequential decisions. Open-ended environment design and the MiniGrid and BabyAI platforms provide complementary settings for generating tasks and studying grounded behavior (Team et al., 2021; Chevalier-Boisvert et al., 2023; 2018). ViGaL studies reasoning through game play, G1 trains perception and reasoning in vision-language models, and SPIRAL uses multi-turn self-play in zero-sum games (Xie et al., 2025; Chen et al., 2025; Liu et al., 2025). Enigmata and Reasoning Gym supply synthetic puzzles and reasoning environments with verifiable outcomes (Chen et al., 2026; Stojanovski et al., 2026). AgentGym-RL studies long-horizon decisions through multi-turn reinforcement learning, while MMSkills develops multimodal skills for visual agents (Xi et al., 2025; Zhang et al., 2026a). RISE generates experience for inferring hidden action rules, revising hypotheses after counterexamples, and reusing the resulting evidence. Its action-only comparison tests the complete supervision format while preserving the source courses and executed trajectories.

Planning, exploration, and teacher supervision. World-model planning and lookahead connect reasoning to action selection in RAP and ProAct (Hao et al., 2023; Yu et al., 2026), while Look Before You Leap studies autonomous exploration for language-model agents (Ye et al., 2026). Expert Iteration combines search with policy learning (Anthony et al., 2017); GKD and MiniLLM study language-model distillation using student-generated sequences (Agarwal et al., 2024; Gu et al., 2024). These approaches motivate distinguishing the capabilities used to produce a decision from the experience exposed to the learner. In RISE, exact symbol decoding and finite program search belong to the solver, while the learning targets record actions and evidence updates from its executed interactions. The audit checks whether a structured record is supported by the public history at that decision, and the exposure controls account for how often each kind of record enters the training loss.

Interfaces and evaluation settings. MineDojo and Voyager study open-ended embodied environments and agents, while Cradle targets general computer control (Fan et al., 2022; Wang et al., 2024; Tan et al., 2025). Game evaluation covers language-model play in lmgame-Bench and TextArena and multimodal interaction in GameWorld (Hu et al., 2025a; Guertler et al., 2025; Ouyang et al., 2026). ALFWorld and WebShop extend interactive evaluation to household tasks and web shopping (Shridhar et al., 2021; Yao et al., 2022). Across these settings, observation and action interfaces define what evidence an agent can obtain and how it can use that evidence. RISE fixes the visual command interface for its matched comparisons and records each external adapter separately, so that a change in interface is distinguishable from a change in learned behavior.

F Environment library and supervision examples

The environment library contains 40 implementations grouped into 32 gameplay families and eight organizational categories. Related implementations share a gameplay family, separating the number of software implementations from the number of distinct gameplay families. All implementations support native generation, visual interaction, reset, and replay through shared interfaces. Training and evaluation use the nine modules listed in the experimental subset, with their task constructors and supervision checks.

Table 18 separates the shared environment interfaces from the procedures used to construct training demonstrations. A generator samples a level specification; its validation routines use the specified dynamics to check transitions and feasible solutions. For the experimental types, a separate solver acts on the public interaction history and supplies source-linked records for the training exporter. The game-specific decoder and planning routines are reused across levels; hypotheses about hidden instance parameters are updated from interaction.

Table 18: Environment and data-generation interfaces. Private level specifications support execution and validation; the demonstration solver receives the public projection.

Procedure	Inputs	Output and role
Environment generator	Game parameters and seed	Level specification, renderer, and transition law.
Task verifier	Level specification and candidate actions	Transition checks and executable solution witnesses for the selected training tasks.
Interaction solver	Public images, feedback, and retained records	Executed actions and source-linked evidence updates; game-specific routines are reused across levels.
Replay and export	Executed interaction and public prefixes	Checked current-command targets and visual training inputs.

Table 19 groups implementations by gameplay family and identifies the experimental subset; Figure 6 shows one rendering per family. The nine experimental types use discrete tasks with independently replayed transitions and solutions, and evidence records checked against public histories. Library interfaces also accommodate dynamic control, partial observations, adversarial interaction, and scheduling tasks. Each generator declares its parameter range; graph rewiring, sequence-machine, and shape-assembly implementations use layout indices from 0 to 15.

Table 20 follows selected commands from executed resource and courier interactions. The resource example records a fuel change after processing; the courier example records vehicle displacement before route planning. Each target cites its source observations, while later commands can select a retained record without asserting an unobserved outcome.

Table 19: Environment-library coverage by gameplay family. Implementation counts include all library variants; the experimental column counts those used for training and evaluation.

Gameplay family	Implementations	Experimental subset
Sliding tiles	1	1
Liquid measure	1	1
Disc towers	1	1
Stack sorting	1	1
Factor sequence	2	0
Shape assembly	1	0
Cellular relay	1	1
Toggle lattice	1	0
Clock circuit	1	0
Chromatic flood	1	0
Graph coloring	1	0
Graph rewiring	1	0
Prism path	1	0
Rooms and keys	2	1
Editable rules	4	1
Balance dispatch	1	0
Recipe workshop	2	1
Deadline scheduler	1	0
Pipeline routing	1	0
Switchyard dispatch	1	0
Inertial dock	1	0
Timed courier	2	1
Projectile dodge	1	0
Pulse intercept	1	0
Falling blocks	1	0
Memory pairs	2	0
Codebreaker	1	0
Mine survey	1	0
Cooperative gates	1	0
Disk capture	1	0
Robot programs	1	0
Tactical charge	1	0
Total	40	9

Table 20: Examples from executed solver interactions. Commands and observations are drawn from exported records; they illustrate data construction rather than learned-model performance.

Game	Command	Observed fact or retained record	Source
Workshop	`act: 5`	Process one input at station A; the remaining fuel permits either candidate processing cost.	`obs_0007`
Workshop	`hypothesize`	Recipe 0 at station A produces material 2; fuel falls from 8 to 7, supporting cost 1.	`obs_0007`, `obs_0008`
Courier	`hypothesize`	The top red vehicle moves from column 2 to column 1; record its displacement before planning the crossing.	`obs_0000`, `obs_0001`
Workshop	`remember`	Retain measured station costs for remaining dependencies and the next level.	`claim_0`



Figure 6: The RISE environment library. Each frame is rendered by one implementation from a gameplay family. Teal markers identify families represented in the nine-game experimental subset; gray markers identify other library families. Related implementations share a family, giving 40 implementations across the 32 displayed families.

F.1 COMPLETED SOLVER TRAJECTORIES

Figures 7 and 8 follow two completed solver courses through selected public observations and commands. The images and outcomes come from recorded executions checked by independent transition replay. These qualitative examples retain their collection-time budgets and are separate from the fixed-budget learner comparisons.



Figure 7: Resource-workshop trajectory. Smelting at furnace B changes fuel from three to two and metal from zero to one. The later level requires eight metal from eight fuel, so the recorded cost guides furnace selection. After building four beacons, the solver reaches the exit and completes both levels. Panels show selected observations in chronological order.



Figure 8: Cellular-relay trajectory. Controls transform the upper binary cells, which must match the lower target. Observations update the solver's evidence, and a reset after loss preserves those records. In the later level, control 5 tests a prediction based on retained observations before subsequent actions reach the new target. The course completes both levels with two resets overall.