EVIDENCE-GUIDED UPDATES FOR REUSABLE AGENT WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents reuse executable workflows to make recurring work efficient, but changing user requirements demand updates that preserve existing responsibilities. Correct examples can identify a new decision without specifying when it should apply, allowing an otherwise successful change to disrupt other uses of the workflow. Starting from an existing program and a small set of correct public execution examples, we propose evidence-guided local updates that combine execution constraints with inherited applicability conditions. The program's conditions are bound across function calls before input adaptation, preserving the behavior of uses unaffected by the new requirement. To reduce the cost of selecting reliable changes, we replay captured decisions to screen bounded candidates and use native execution to accept the selected program. On authored reconciliation workflows, inherited conditions improve evidence-registration success from 60% to 100%. For identical final programs, replay reduces incremental acquisition time by 61.7% relative to native interpreter reuse with early stopping.

1 INTRODUCTION

An assistant handling recurring work must adapt when its user changes which document versions count, which suppliers are equivalent, or who may approve a request. Reusable programs encode these conventions and save repeated reasoning. Their next update should implement the new requirement while preserving the workflow's other responsibilities. A user expects the assistant to incorporate the changed rule without having to restate every convention that still applies.

Workflow induction and programmatic skills let agents reuse procedures acquired from experience (Wang et al., 2025b;a; 2024a), while automated workflow search explores how those procedures can be organized (Zhang et al., 2025; Hu et al., 2025). Once a program serves several purposes, updating it requires resolving two linked questions: which changes fit the correct executions, and which situations permit those changes? A successful teaching trace may constrain the first while leaving the second unspecified.

Consider a supplier alias that should affect receipt matching while evidence registration keeps the original identity (Figure 1). Normalizing the input before a shared decision call can change both uses, even if the downstream evidence condition remains in the source. Such updates can fit correct operational examples. Across three relational updates, both variants pass all teaching, processing, and audit checks, but evidence-registration success falls from 75/75 to 45/75 when the inherited condition is removed. Preserving code alone therefore does not preserve the meaning of its inputs.

The deployed program offers information that examples alone may miss. Its conditions identify uses that should retain their original inputs. The decision boundary also separates a proposed change from the application interactions around it, creating an opportunity to test alternatives without repeatedly executing the full workflow.

Our first contribution combines public execution constraints with inherited applicability conditions. Public reports constrain the desired decision; conditions bound to the caller's arguments determine when inputs may change. Evaluating those conditions before adaptation preserves the meaning of unaffected branches. Eligible original returns protect already settled decisions, while recorded alternatives allow earlier rules to be refined without discarding later transformations.

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.

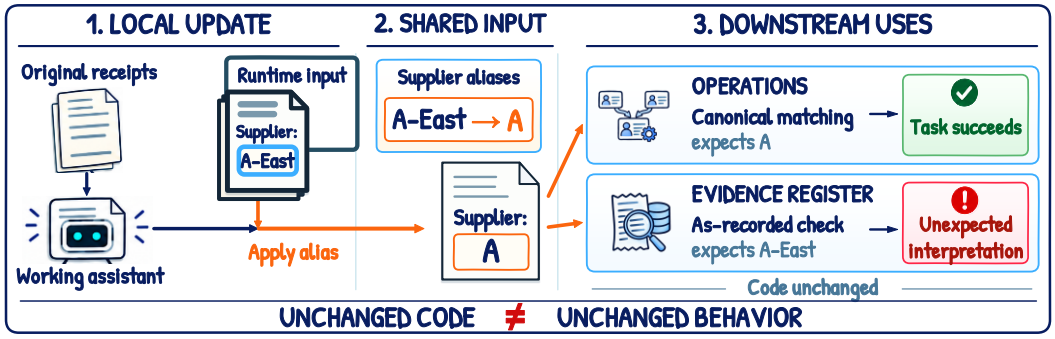

Figure 1: **A local update can affect other workflow uses.** An unrestricted supplier-alias update maps the shared runtime input from `A-East` to `A`. Both uses receive `A`: receipt matching succeeds, whereas evidence registration requires the as-recorded `A-East`. Original receipts and downstream code remain unchanged.

Our second contribution reduces acquisition cost by validating candidates at that decision boundary. We capture decision inputs during teaching execution, screen bounded candidates against public targets by pure replay, and run the selected program in the native environment for acceptance. This retains the application-level delivery check while reducing repeated environment execution. Comparisons with interpreter reuse and early stopping select identical programs using fewer teaching worlds.

We evaluate these contributions on authored reconciliation and maintenance workflows through task completion, preservation of original uses, and incremental acquisition cost. Strong editing and selection controls and component interventions establish the roles of conditions and replay. ALFWorld reuse and THOR/Minecraft component studies provide complementary evidence about native execution.

2 Problem Setup

The learner receives an existing program P, called the parent, and public material D: a change request and correct executions containing actions, observations, and delivered outputs. Revisiting a specified earlier update also supplies its recorded alternatives and edit location. The returned program P^+ persists across tasks and updates; model weights remain fixed. Evaluation uses fresh contexts and specified world states.

Delivery and unchanged responsibilities. An update should implement the new convention and preserve each workflow use. Success checks outputs and final application state, including effects that must remain unchanged. We test previously successful tasks under the same controller and use-specific behavior on new inputs.

Incremental acquisition cost. Given the parent and supplied executions, we record

$$\mathbf{C}_{\Delta} = \mathbf{C}_{\text{propose}} + \mathbf{C}_{\text{capture}} + \mathbf{C}_{\text{select}} + \mathbf{C}_{\text{feedback}} + \mathbf{C}_{\text{accept}} + \mathbf{C}_{\text{repair}}. \tag{1}$$

The terms cover construction or generation, capture, screening, additional teaching, acceptance, and repair. Execution in the application environment is *native execution*. Costs include tokens, model time, API calls, and enclosing execution time, including failures. Historical acquisition, shared teaching, service startup, reuse, and research evaluation are recorded separately.

3 Updating Decisions with Execution Evidence

The method separates what a new convention changes from when it applies. Public execution outputs constrain changed decisions; conditions inherited from the parent protect its other uses (Figure 2).

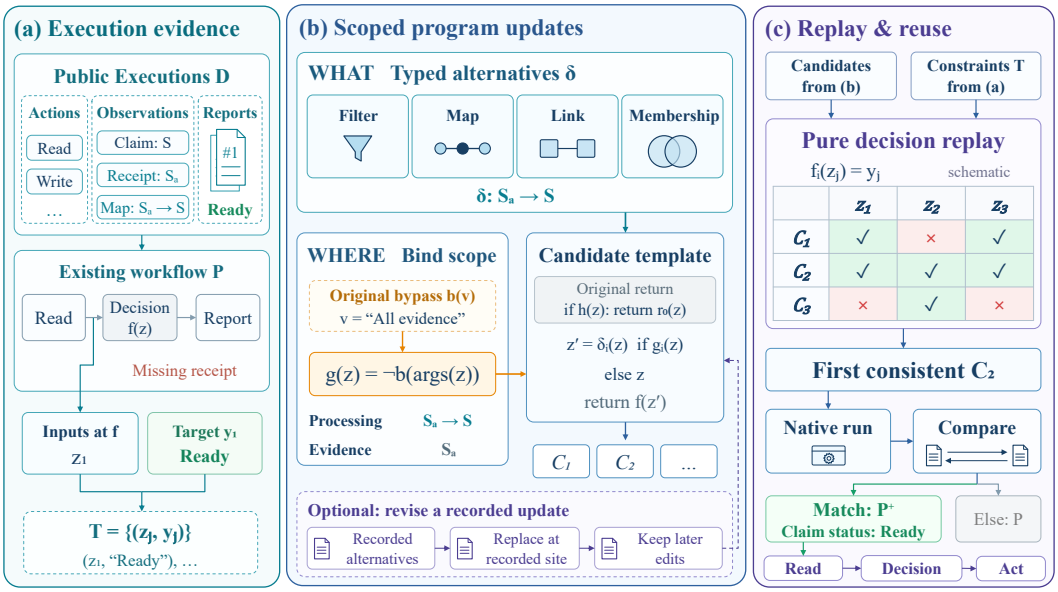

Figure 2: **Framework for evidence-guided workflow updates.** (a) Executions of the parent P capture decision inputs z_j; correct public reports supply aligned targets y_j. (b) Typed adapters and bound scope conditions construct candidate programs. The gray prefix evaluates an eligible original return before changing inputs; the optional lower branch revises a recorded subtree while preserving later edits. (c) Pure decision replay selects the first consistent candidate for native report acceptance and workflow reuse. S_a and S are the alias and canonical identities (`A-East` and `A` in Figure 1); v is the receipt view, and h and r_0 are the original return condition and expression. The displayed template illustrates an entry adaptation; scalar adapters may modify internal call arguments. Candidate checks are schematic.

3.1 Capture the decision constrained by an execution

We extract a directly assigned decision call inside a record loop and its pure helper dependencies. Structural checks exclude application APIs, reassignment of protected inputs, and output state feeding later decisions. Replay uses fresh argument copies and rejects mutation; application effects remain in the surrounding workflow.

Executing the instrumented parent in teaching worlds captures arguments z_j. Shared identity fields align records with rows in its report; a field matching the decision's return identifies the output to constrain. The corresponding row in the correct report supplies y_j. Alignment must be unambiguous and cover all reported records, giving $T = \{(z_j, y_j)\}$ from public outputs without the teacher's implementation or private task grader.

In the supplier-alias example in Figure 2(a), z_1 contains a claim under supplier S, a receipt under alias S_a, and their supplied map. The parent returns *Missing receipt*, while the correct public report supplies the claim-status target $y_1 =$ Ready.

3.2 Construct bounded changes to decision inputs

An adapter changes a decision input within a finite program space (Solar-Lezama et al., 2006). Scalar adapters filter a collection argument at an internal call using type-compatible comparisons between record and context fields. Missing context preserves the argument; missing comparison metadata preserves the record.

Relational adapters enumerate string-map normalization, identifier-linked lookup, and set-membership Boolean enrichment. Alternatives vary mapping direction, lookup precedence, and Boolean replacement or combination. Captured types and overlap with relation keys or values bind

fields. Targets test candidates; they do not bind fields. Enumeration removes duplicate structures and uses a fixed budget.

For an entry adaptation, let f be the original decision, δ the input change, and g its application condition:

$$f_{\delta,g}(z) = \begin{cases} f(\delta(z)), & g(z), \\ f(z), & \text{otherwise.} \end{cases} \tag{2}$$

Scalar adapters change an internal call argument; replay evaluates the enclosing decision.

3.3 Bind conditions before adapting inputs

We extract the callee's top-level conditional returns whose conditions depend on bound parameters other than the leading data argument. Substituting positional arguments, keywords, and defaults expresses each condition at the caller. Eligible expressions depend only on decision parameters. Their disjunction defines the bypass condition; its negation supplies g. Scalar adapters also retain their missing-context bypass.

The WHERE branch in Figure 2(b) illustrates the receipt-view condition $b(v)$, which holds for the *All evidence* view. Here $\mathrm{args}(z)$ binds v to the value supplied by the caller, giving $g(z) = \neg b(\mathrm{args}(z))$ before adaptation.

An original terminal return can protect an already settled decision. We preserve an eligible entry prefix of pure local assignments and an initial conditional return. In the figure's protected template, $h(z)$ and $r_0(z)$ denote that condition and return expression, evaluated on the original inputs before adaptation. Thus linked pending state need not overwrite a locally settled outcome. Protected and unprotected variants remain candidates. These conditions define an adapter's *scope*; inherited-use variants precede unrestricted variants in the fixed order.

3.4 Replay, accept, and refine recorded updates

Let $\mathcal{A}(P, D)$ be the ordered candidate programs and f_p their decision routines. For its ith candidate C_i, Figure 2(c) abbreviates f_{C_i} as f_i. Public constraints define

$$\mathcal{V}(T) = \{p \in \mathcal{A}(P, D) : f_p(z_j) = y_j \text{ for all } (z_j, y_j) \in T\}. \tag{3}$$

An exception, mutation, or target mismatch rejects a candidate. The first survivor undergoes native teaching execution; public reports must match before installation. An already matching parent, empty selection, or rejected update retains the parent. Evaluation follows teaching-only selection.

For a specified historical update, the lower branch of Figure 2(b) replaces exact matching syntax subtrees at its recorded call argument with a recorded alternative, preserving later enclosing transformations and other functions. The historical site and expression must remain identifiable. Our composition study crosses three earlier teaching-consistent alternatives with two return-protection variants and selects using two additional correct executions.

3.5 Optional feedback for ambiguous updates

Typed edits to captured teaching inputs can distinguish surviving candidates (Seung et al., 1992). Uniform weights and fixed tie-breaking select the edit with highest prediction entropy; a teacher executes that world and supplies its public trace and report. Queries stop at a unique candidate, an indistinguishable pool, or the budget limit. Planning, construction, teaching, capture, and acceptance are charged. This query extension is evaluated separately from the recorded-update branch in Figure 2(b), which uses supplied ordinary executions.

4 Experimental setup

Tasks and evaluation. Business reconciliation uses AppWorld APIs (Trivedi et al., 2024). Six updates cover effective dates, receipt quality, revision snapshots, supplier aliases, settlement links, and delegated approval. Each relation uses two correct teaching executions and fifteen processing, audit, or evidence-registration tasks; nineteen original tasks measure retention. An authored maintenance

Table 1: Reliable updates across three relational changes. Each cell reports success rate (%) with its numerator/denominator, or mean incremental acquisition time.

	Supplier alias		Settlement link		Delegated approval	
Update	Success (%)	Time (s)	Success (%)	Time (s)	Success (%)	Time (s)
Parent (no update)	73.3 (11/15)	–	60.0 (9/15)	–	73.3 (11/15)	–
Full rewrite	86.7 (39/45)	367.40	73.3 (33/45)	366.36	80.0 (36/45)	385.85
Function edit	86.7 (39/45)	57.73	80.0 (36/45)	143.43	80.0 (36/45)	65.15
Short choice	91.1 (41/45)	35.11	93.3 (42/45)	35.32	97.8 (44/45)	37.90
Thinking choice	100.0 (45/45)	479.37	86.7 (39/45)	632.75	100.0 (45/45)	597.48
Decision replay	100.0 (15/15)	11.70	100.0 (15/15)	10.94	100.0 (15/15)	12.22

Generated updates use three draws on the same 15 tasks per event; replay updates the same existing program deterministically. All retain 19/19 original tasks per draw.

workflow uses file and Todoist APIs with two provided program structures. Success requires correct reports and final application state; task views are counted separately from independent batches.

Comparisons and repetition. Qwen3.5-27B (Qwen Team, 2026) supplies full-program rewriting, function editing, and plain or thinking candidate selection; selectors receive the same options as replay. Earlier studies also use Qwen3-8B (Yang et al., 2025) and Granite-3.3-8B (IBM Granite, 2025). AWM (Wang et al., 2025b) and SkillCraft (Chen et al., 2026) adaptations retain workflow induction/retrieval or executable skill acquisition/composition. Relational controls use three acquisition seeds on the same fifteen tasks; maintenance and composition controls use two seeds per program/history. Five parents come from four acquisition draws. Selection uses public teaching; evaluation follows fixed artifacts in fresh worlds.

Cost and environmental roles. Incremental acquisition includes generation, format attempts, capture, construction, reception, checking, and native acceptance; requested feedback also charges teaching. We report model time, tokens, APIs, and enclosing update time, with historical resources, shared teaching, service startup, and research evaluation separately recorded. ALFWorld tests program reuse. THOR and Minecraft test native execution with matched controller privileges. Policies receive structured observations; rendered frames show their resulting behavior.

5 Results

5.1 Reliable relational updates with lower acquisition cost

Across five parents, alias and delegated-approval success each rises from 55/75 to 75/75; settlement linkage rises from 45/75 to 75/75. Each event preserves 95/95 original-task evaluations. The updates match recognized aliases, inherit previously settled state, and recognize authorized delegates.

Thinking selection reaches 15/15 in every alias and delegated-approval acquisition. Replay matches this quality for 11.70 and 12.22 mean incremental seconds, versus 479.37 and 597.48 seconds (Table 1; Figure 3). Settlement selection reaches 42/45 without thinking and 39/45 with it.

Quality and revision reach 15/15 after updating, versus 9/15 with AWM 27B online reasoning and 9/15 with direct SkillCraft libraries. SkillCraft with the same editable decision routine remains at 9/15 on both, saving no new skill. Table 10 and Appendix E report all eight online conditions and costs, distinguishing direct execution from acquiring the changed decision.

5.2 Inherited conditions preserve the intended use

Application conditions protect uses that teaching examples leave ambiguous. On supplier aliases, ordinary function and full edits each solve all operational and audit tasks but only 9/15 evidence-registration tasks, retaining 57/57 original checks. Normalization alters evidence inputs despite unchanged consumer code.

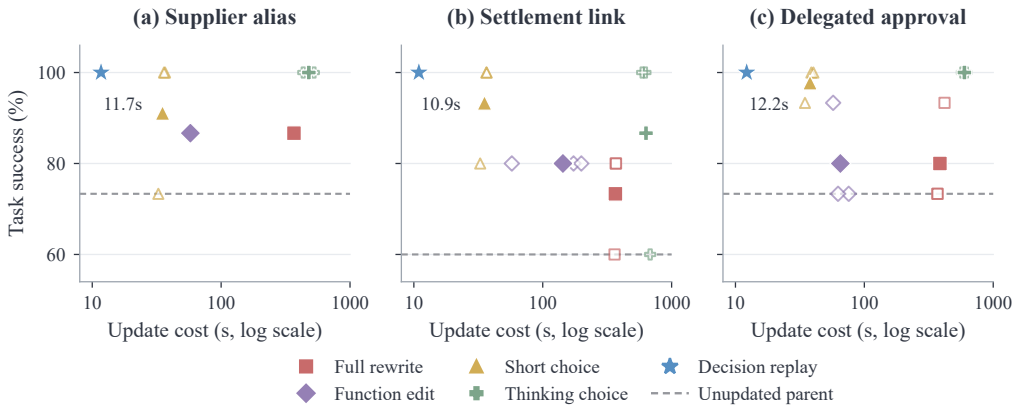

Figure 3: Quality and acquisition cost across three updates. Hollow marks show three generation draws on the same fifteen tasks; filled marks show their means. Replay is a deterministic given-parent result. Dashed lines indicate parent quality. The logarithmic cost axis measures complete incremental acquisition; historical resources are reported separately.

Removing only the inherited condition lowers evidence registration from 75/75 to 45/75 (Figure 4). With parent, operator, and fields fixed, both variants pass 30/30 teaching and all 75 operational and 75 audit evaluations in this exploratory intervention.

5.3 Replay reduces validation for the same final program

All validation strategies select identical programs across twenty updates from five parents and two quality–revision orders. Cold native search averages 308.04 learning seconds and 64 teaching worlds. Interpreter reuse takes 110.40 seconds; first-counterexample stopping takes 57.52 seconds and 33 worlds. Replay uses four worlds and 22.04 seconds, a 61.7% reduction relative to that optimized control.

Including recorded parent acquisition and both updates gives 343.74 versus 414.71 seconds, a 17.1% reduction. For alias and settlement, replay halves optimized native validation from eight worlds to four. Delegated approval's first candidate succeeds, leaving both strategies at four worlds and similar time.

5.4 Refining earlier decisions improves continued use

Ten quality–revision or revision–quality prefixes each acquire three further relational updates from their preceding installed program. All thirty stages reach 15/15, retaining earlier successes in 2,448 repeated checks. Three added updates average 27.58 seconds; recorded parent, prefix, and added costs average 371.31 seconds. Prefix and continuation orders are paired (Appendix A, Figure 7).

Frozen relational programs reach 30/30 new views per event. On simultaneous-rule tests, continued programs reach 10/18 versus 6/18 before the relations. Mixed relations succeed; a proxy revision field and linked pending state overwriting locally settled state explain the remaining gaps (Appendix F).

A separate follow-up combines historical revision refinement and protection of the original return, reaching 36/36 new instances under both histories using two additional correct public executions. Either component alone raises 20/36 to 28/36. When both interactions occur in one batch, either component reaches 3/9 and the combination 9/9. Both combined programs retain 19/19 original tasks (Table 2).

A post-hoc comparison fixes candidates and selected programs: replay reduces native worlds from ten to four and API calls from 652 to 256. Replay takes about 10.5 seconds versus 22.7 for warm native early stopping, saving 53.6%.

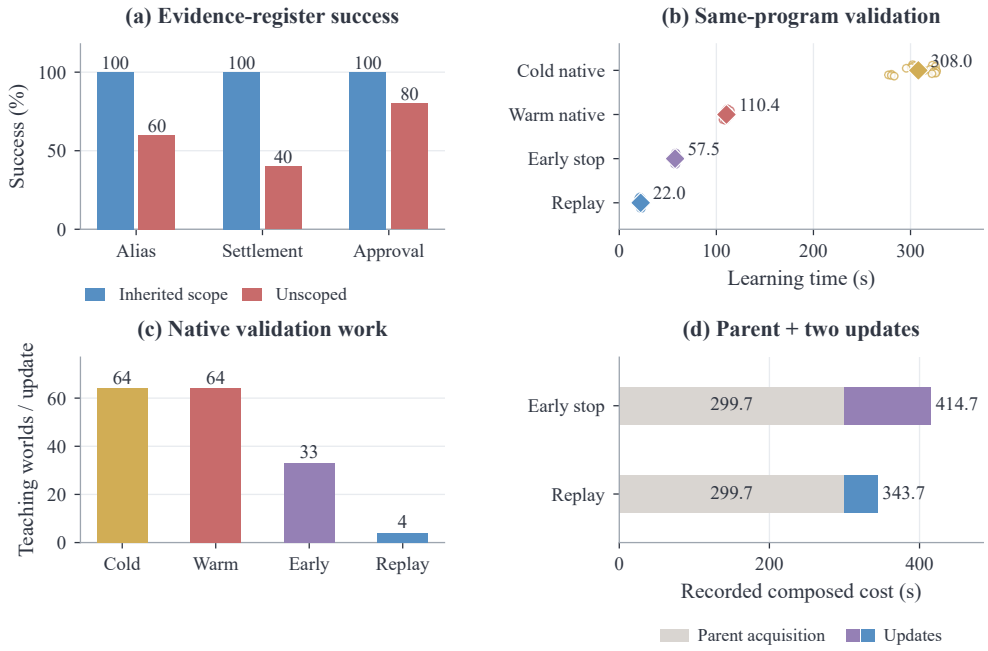

Figure 4: Inherited scope protects uses; replay reduces validation cost. (a) Evidence registration across three events; both scope settings also pass all teaching, operational, and audit checks. (b) Circles show twenty measured update stages; diamonds show means. All four validation methods select identical programs. (c) Native teaching worlds per update. (d) Mean recorded parent acquisition plus two updates across ten streams. Repeated stages share five parent resources.

Table 2: Historical-rule refinement and return protection are complementary. (a) Component scores are identical under both histories. (b) All acquisition outcomes, including rejected proposals and their fallback deployments.

(a) Component intervention

Update	New views	Joint views
Original combination	20/36	3/9
Rule refinement only	28/36	3/9
Return protection only	28/36	3/9
Combined update	36/36	9/9

(b) Acquisition and deployed quality

Method	Accepted	New (%)	Joint (%)	Update (s)
Function edit (plain)	0/4	55.6	33.3	115.62
Thinking choice	1/4	66.7	50.0	1127.42
Combined update	2/2	100.0	100.0	10.52

Model controls use two seeds per history; the combined update is deterministic on each history. Percentages average the same 36/9-view pools. All deployments retain 19/19 original tasks. Times average complete incremental acquisition, including failures; shared resources are separate. Appendix H lists each condition.

One of eight model acquisitions matches all outcomes: thinking selection costs 829.87 incremental seconds versus 10.53. The seven rejected or length-limited proposals retain the parent. Recorded history and shared teaching give matched path sums of 1322.18 and 502.85 seconds. Table 2 marks decoding modes; Appendix H separates service and research costs.

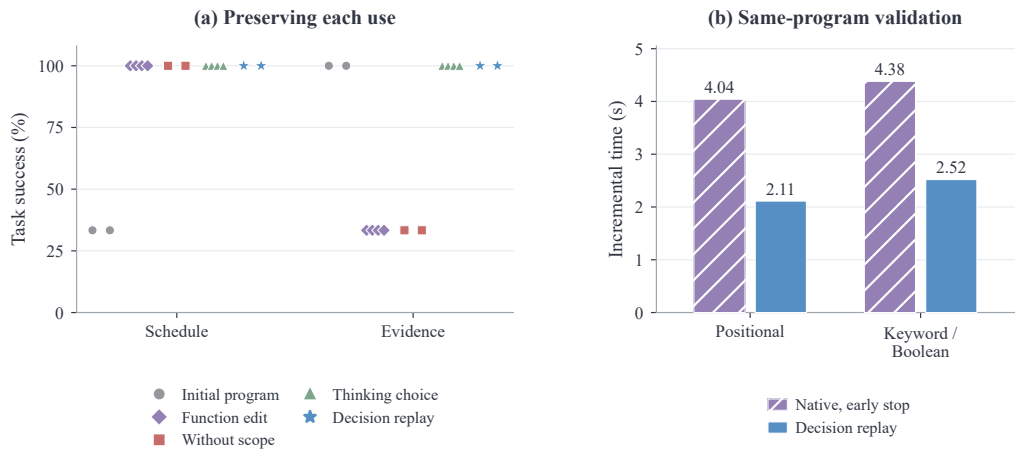

Figure 5: Directory-identity updates preserve each maintenance use. (a) Every point is a provided structure or model draw on the same six scenarios in two uses. Both model controls use thinking. Function edits solve scheduling but change evidence behavior. (b) Replay and warm native early stopping select identical programs; costs include capture and construction.



Figure 6: Same-world Minecraft comparison for crafting a wooden pickaxe. Circles mark workbench regions in original Prismarine frames at capture +25 s. Outcomes follow saved-game task checks. Calls count public API requests, including failures. State access reuses the table, while full rewriting creates another. Original-speed recordings and complete traces are included.

5.5 Identity-map updates transfer to maintenance scheduling

Identity-map adaptation and condition binding transfer to maintenance scheduling: both calling structures improve from 8/12 to 12/12 views across six scenarios, retaining time-window, occupancy, and capacity logic. Four thinking-based function edits solve scheduling but alter evidence use, each reaching 8/12. Four thinking selectors match 12/12. Removing the condition also returns to 8/12 (Figure 5).

For the same final programs, replay takes 2.11/2.52 seconds versus 4.04/4.38 for warm native early stopping, halving worlds and APIs. Both controls use thinking on a shared GPU; Appendix G gives each draw and cost component.

Appendix A evaluates targeted feedback for updates left ambiguous by the initial examples.

5.6 STANDARD REUSE AND VISIBLE NATIVE EXECUTION

On standard ALFWorld valid-unseen tasks (Shridhar et al., 2021), the deployed parent and a public-interface dispatcher each solve 60/60; replacing the program with its THOR update yields 43/60. Model-driven controls using all twelve demonstrations or BM25 top-two reach 59/60 with their respective resources.

In AI2-THOR (Kolve et al., 2017), corrected observation decoding, an experience-derived camera sequence, and per-action goal checks succeed in 7/12 rooms in each of two new-room pools. A short selector also reaches 7/12 on the first pool. All share visible-object IDs, oracle navigation, and `forceAction`.

In Minecraft, state-access correction and full rewriting both deliver a pickaxe from the same world using twenty and twenty-two APIs; the draft fails. Figures 6 and 8 and original-speed videos show these outcomes.

6 RELATED WORK

Reusable procedures. Toolformer learns API use (Schick et al., 2023). ReAct couples reasoning with actions; Reflexion and ExpeL reuse experience (Yao et al., 2023; Shinn et al., 2023; Zhao et al., 2024). DreamCoder and LILO learn program libraries; Parsel composes functions under execution constraints (Ellis et al., 2021; Grand et al., 2024; Zelikman et al., 2023). Workflow and skill-induction studies address reusable procedures (Wang et al., 2025b; 2024b; 2025a; Zheng et al., 2025; Chen et al., 2026). Programs also support embodied control (Liang et al., 2023; Wang et al., 2024a).

Program refinement and search. Self-Refine revises outputs using model feedback; Self-Debugging and AdaPlanner use code explanations or execution feedback (Madaan et al., 2023; Chen et al., 2024; Sun et al., 2023). AFlow, ADAS, and AgentSquare search workflow code or agent components (Zhang et al., 2025; Hu et al., 2025; Shang et al., 2025). We update existing programs while preserving their other uses.

Constraints and validation. ContractSkill repairs skills with contracts; NSI induces logic and bindings from traces (Lu et al., 2026; Shao et al., 2026). LEVER learns execution-based verification (Ni et al., 2023). Constrained completion, semantic repair, regression-test selection, and test-equivalence analysis provide construction and validation foundations (Solar-Lezama et al., 2006; Mechtaev et al., 2016; Rothermel & Harrold, 1997; Mechtaev et al., 2018).

7 DISCUSSION AND CONCLUSION

Correct executions and inherited conditions provide complementary evidence for updating reusable workflows. Public outputs constrain the desired decision, while conditions bound before input adaptation constrain where the change applies. The existing program thus contributes information about unaffected uses that successful examples alone may leave unspecified. At the same decision boundary, replay reuses captured evidence across candidates, reducing repeated workflow execution while retaining native acceptance. Recorded alternatives also allow earlier rules to be refined without discarding later transformations, helping workflows accommodate subsequent requirements.

Our evaluation covers two authored workflows, existing programs, and pure decision boundaries. Historical refinement requires identifiable recorded expressions; the embodied studies examine reuse and execution components. Broader natural workflows and arbitrary application acquisition remain open.

We presented evidence-guided local updates that combine public execution constraints with inherited applicability conditions. Across the studied workflows, these updates preserve unaffected uses, while decision replay reduces validation cost for identical final programs. Existing workflows thus supply not only reusable behavior, but also conditions for reliable adaptation.

AI USE STATEMENT

AI assistants supported research exploration, experiment design, implementation, analysis, literature checks, and drafting text and figures. Experimental model calls are recorded separately from this assistance. Quantitative graphics derive from archived results; environment images use original rendered frames with explanatory annotations; the introductory example and method overview are conceptual illustrations.

REPRODUCIBILITY STATEMENT

The supplement includes executable candidate generation and decision replay with archived public constraints, scripts for rebuilding principal result tables, and original THOR/Minecraft recordings with action traces and native outcomes. The appendix specifies tasks, models, and cost conventions. Full native reruns require the documented simulator and model assets. Acquisition and queries use declared learning tasks; each study evaluates fixed updates, and diagnostic findings inform separately defined follow-ups.

REFERENCES

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. SkillCraft: Can LLM agents learn to use tools skillfully?, 2026. URL https://arxiv.org/abs/2603.00718.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, pp. 8746–8825, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2460396f2d0d421885997dd1612ac56b-Abstract-Conference.html.

Kevin Ellis, Lionel 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. Association for Computing Machinery, 2021. doi: 10.1145/3453483.3454080. URL https://doi.org/10.1145/3453483.3454080.

Gabriel Grand, Lio Wong, Maddy Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations*, pp. 30399–30446, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/819cebb05f993840e8a52d7564c5c282-Abstract-Conference.html.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

IBM Granite. Granite 3.3 8B Instruct model card. Hugging Face model card, 2025. URL https://huggingface.co/ibm-granite/granite-3.3-8b-instruct.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI, 2017. URL https://arxiv.org/abs/1712.05474.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://ieeexplore.ieee.org/document/10160591.

Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. ContractSkill: Repairable contract-based skills for multimodal web agents, 2026. URL https://arxiv.org/abs/2603.20340.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In *Proceedings of the 38th International Conference on Software Engineering*, pp. 691–701, 2016. doi: 10.1145/2884781.2884807. URL https://discovery.ucl.ac.uk/id/eprint/10088929/.

Sergey Mechtaev, Xiang Gao, Shin Hwei Tan, and Abhik Roychoudhury. Test-equivalence analysis for automatic patch generation. *ACM Transactions on Software Engineering and Methodology*, 27(4):1–37, 2018. doi: 10.1145/3241980. URL https://mechtaev.com/files/tosem18.pdf. Article 15.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26106–26128. PMLR, 2023. URL https://proceedings.mlr.press/v202/ni23b.html.

Qwen Team. Qwen3.5-27B model card. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-27B.

Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, 1997. doi: 10.1145/248233.248262. URL https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/p173-rothermel.pdf.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 287–294, 1992. doi: 10.1145/130385.130417. URL https://collaborate.princeton.edu/en/publications/query-by-committee/.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *International Conference on Learning Representations*, pp. 3841–3865, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0ae94013da7cd459402fd77874e09ee3-Abstract-Conference.html.

Jie-Jing Shao, Haiyan Yin, Yueming Lyu, Xingrui Yu, Lan-Zhe Guo, Ivor W. Tsang, James T. Kwok, and Yu-Feng Li. Lifting traces to logic: Programmatic skill induction with neuro-symbolic learning for long-horizon agentic tasks, 2026. URL https://arxiv.org/abs/2605.01293.

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*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 404–415, 2006. doi: 10.1145/1168857.1168907. URL https://people.eecs.berkeley.edu/~sseshia/pubs/b2hd-solar-asplos06.html.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 58202–58245, 2023. doi: 10.52202/075280-2537. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b5c8c1c117618267944b2617add0a766-Abstract-Conference.html.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

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. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zhiruo Wang, Graham Neubig, and Daniel Fried. TroVE: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51177–51191. PMLR, 2024b. URL https://proceedings.mlr.press/v235/wang24az.html.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Conference on Language Modeling*, 2025a. URL https://arxiv.org/abs/2504.06821.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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. URL https://arxiv.org/abs/2210.03629.

Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions. In *Advances in Neural Information Processing Systems*, volume 36, pp. 31466–31523, 2023. doi: 10.52202/075280-1367. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6445dd88ebb9a6a3afa0b126ad87fe41-Abstract-Conference.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, pp. 34040–34077, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills, 2025. URL https://arxiv.org/abs/2504.07079.

A CONTINUED UPDATES AND TARGETED FEEDBACK

Figure 7 shows continued updates from Section 5.4 and the independent feedback study below. Continued streams reuse the preceding installed program; feedback queries resolve ambiguity among candidate updates.

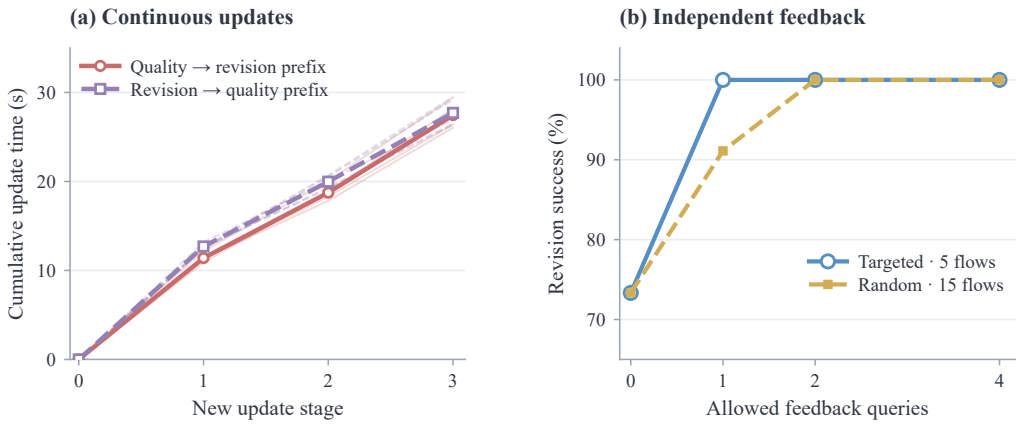

Figure 7: Continuous updates and independent feedback. (a) Faint lines show ten continuations; thick lines average five parents per prefix order. Previously successful tasks pass all 2,448 repeated checks. (b) With one query, all five targeted flows and ten of fifteen random flows solve every revision task. Budgets use prefixes of the same feedback streams. Quality admission remains 15/15 at every budget. Feedback is evaluated independently of the continuations.

A.1 ADDITIONAL FEEDBACK HELPS WHEN AMBIGUITY AFFECTS DECISIONS

In a separate five-parent study with shifted metadata, one targeted execution raises revision success from 11/15 to 15/15; one random execution averages 13.67/15. Random querying matches this quality with 1.33 executions on average. Initial update plus feedback costs 50.01 versus 55.81 seconds. Quality admission already succeeds at 15/15: targeted feedback identifies a unique rule with one execution, versus 2.33 at random, without a quality gain. Budgets 0/1/2/4 are nested prefixes of the same streams (Figure 7); this query study is independent of continued-use refinement.

A.2 NATIVE THOR EXECUTION

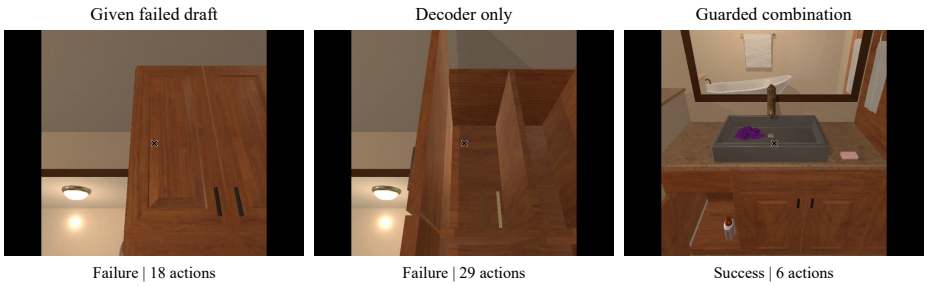

Figure 8: Observation decoding and closed-loop action checks affect native THOR delivery. In the same teaching scene, the draft and decoder-only variants fail after 18 and 29 actions, while the guarded combination succeeds in 6. Original Unity frames and native goal checks establish these outcomes. The supplementary comparison retains all three original-speed recordings.

B RELATIONAL UPDATE PROTOCOLS AND INDIVIDUAL RESULTS

Data and notation. Teaching executions provide the learner with successful public traces and reports; model weights remain fixed. Evaluation tasks measure the selected program in fresh worlds. Tables use *new* for the changed task and *old* for the original tasks; retention counts only task identities the parent already solves. P1–P5 identify five distinct parent programs from four original acquisition draws. Task views, repeated model draws, and consecutive stages reuse task identities, so their summed denominators count task executions rather than independent tasks.

The three relational changes are supplier-identity normalization, lookup of prior settlement through an identifier relation, and delegated authority through set membership. Each event provides two teaching executions and fifteen evaluation tasks: five case constructions viewed through operational processing, audit, and evidence registration. The generators vary identifiers, composition with earlier rules, mismatched evidence, missing context, and legacy metadata. Parent qualification uses only the original teaching tasks.

Table 3: Relational updates across five parent resources. Replay and native early stopping select the same candidate in every pair.

Event	Parent	Before	After	Original	Replay (s)	Early stop (s)	Executions Replay/Early
Alias	P1	11/15	15/15	19/19	11.70	17.06	4/8
Alias	P2	11/15	15/15	19/19	10.82	16.77	4/8
Alias	P3	11/15	15/15	19/19	10.55	16.48	4/8
Alias	P4	11/15	15/15	19/19	10.65	16.23	4/8
Alias	P5	11/15	15/15	19/19	11.91	17.80	4/8
Settlement	P1	9/15	15/15	19/19	10.94	19.08	4/8
Settlement	P2	9/15	15/15	19/19	10.90	16.31	4/8
Settlement	P3	9/15	15/15	19/19	10.10	16.04	4/8
Settlement	P4	9/15	15/15	19/19	10.55	16.00	4/8
Settlement	P5	9/15	15/15	19/19	10.35	16.18	4/8
Approval	P1	11/15	15/15	19/19	12.22	11.46	4/4
Approval	P2	11/15	15/15	19/19	11.88	11.39	4/4
Approval	P3	11/15	15/15	19/19	11.63	11.11	4/4
Approval	P4	11/15	15/15	19/19	11.34	11.19	4/4
Approval	P5	11/15	15/15	19/19	11.87	11.44	4/4

P1–P5 are five programs from four acquisition draws. Execution counts include the native teaching checks required by each procedure. R/E denotes replay/early stopping.

Table 4: Supplier alias: individual acquisition outcomes.

Method	Seed	Accepted	New	Original	Δs	Model s	Input tok.	Output tok.
Full rewrite	1729	Yes	13/15	19/19	370.37	366.48	35742	4429
Full rewrite	2718	Yes	13/15	19/19	365.77	361.92	35742	4349
Full rewrite	3141	Yes	13/15	19/19	366.06	361.95	35742	4365
Function edit	1729	Yes	13/15	19/19	57.93	54.40	35767	448
Function edit	2718	Yes	13/15	19/19	56.55	52.75	35767	442
Function edit	3141	Yes	13/15	19/19	58.70	54.78	35767	460
Plain choice	1729	Yes	15/15	19/19	36.74	21.34	42177	7
Plain choice	2718	Yes	15/15	19/19	36.01	20.65	42177	7
Plain choice	3141	No	11/15	19/19	32.57	20.90	42178	7
Thinking choice	1729	Yes	15/15	19/19	479.51	463.58	42175	5594
Thinking choice	2718	Yes	15/15	19/19	524.51	508.60	42175	6118
Thinking choice	3141	Yes	15/15	19/19	434.09	418.19	42176	4975

All draws share 15 event tasks and 19 original tasks. Thinking output includes reasoning and the final answer.

Generative controls. Full-program and function proposals receive the complete parent source and the same correct public executions. Candidate selectors also receive the candidate pool and captured decision arguments; their costs include capture and candidate construction. Candidate

Table 5: Settlement link: individual acquisition outcomes.

Method	Seed	Accepted	New	Original	Δs	Model s	Input tok.	Output tok.
Full rewrite	1729	Yes	12/15	19/19	367.85	363.88	36644	4387
Full rewrite	2718	No	9/15	19/19	360.74	358.03	36644	4279
Full rewrite	3141	Yes	12/15	19/19	370.48	366.69	36644	4388
Function edit	1729	Yes	12/15	19/19	57.58	54.32	36669	454
Function edit	2718	Yes	12/15	19/19	173.37	169.51	36669	1913
Function edit	3141	Yes	12/15	19/19	199.35	195.55	36669	2175
Plain choice	1729	Yes	15/15	19/19	36.52	21.64	43905	7
Plain choice	2718	Yes	12/15	19/19	32.72	21.74	43905	7
Plain choice	3141	Yes	15/15	19/19	36.74	21.87	43906	7
Thinking choice	1729	No	9/15	19/19	681.33	666.97	43903	8014
Thinking choice	2718	Yes	15/15	19/19	625.79	615.23	43903	7324
Thinking choice	3141	Yes	15/15	19/19	591.12	580.55	43904	6957

All draws share 15 event tasks and 19 original tasks. Thinking output includes reasoning and the final answer.

Table 6: Delegated approval: individual acquisition outcomes.

Method	Seed	Accepted	New	Original	Δs	Model s	Input tok.	Output tok.
Full rewrite	1729	No	11/15	19/19	368.40	365.15	35759	4434
Full rewrite	2718	No	11/15	19/19	370.25	366.57	35759	4436
Full rewrite	3141	Yes	14/15	19/19	418.92	415.61	35759	5047
Function edit	1729	Yes	14/15	19/19	57.27	53.65	35784	461
Function edit	2718	No	11/15	19/19	75.64	72.29	35784	683
Function edit	3141	No	11/15	19/19	62.54	59.26	35784	520
Plain choice	1729	Yes	15/15	19/19	38.81	22.44	44422	8
Plain choice	2718	Yes	14/15	19/19	34.63	23.04	44422	8
Plain choice	3141	Yes	15/15	19/19	40.25	23.87	44423	7
Thinking choice	1729	Yes	15/15	19/19	602.78	587.59	44420	7030
Thinking choice	2718	Yes	15/15	19/19	618.00	602.82	44420	7230
Thinking choice	3141	Yes	15/15	19/19	571.67	556.48	44421	6648

All draws share 15 event tasks and 19 original tasks. Thinking output includes reasoning and the final answer.

order is shuffled by acquisition seed. Thinking and plain selectors receive the same messages and options.

All relational generative controls use seeds 1729, 2718, and 3141. Adapter enumeration is capped at 128 candidates in its fixed order. Full-program and function proposals have a 16,384-token output cap and at most two format attempts. Plain selection uses one request capped at 1,024 tokens; thinking selection uses one request capped at 16,384 tokens, counting reasoning and answer tokens together. Inputs fit within 55,000 tokens without truncation. A proposal that fails public acceptance leaves the parent deployed; the tables include these outcomes and the costs of all attempts.

C Application conditions and successive updates

Application-condition intervention. Each selected relational adapter is paired with a variant applying the same operation in every mode. Operator, fields, mapping direction, precedence, and Boolean combination are fixed; only the outer application condition changes. Both variants satisfy the public teaching executions. This exploratory intervention follows the observed differences among consumer views in ordinary updates.

Successive updates. Ten continuations start from previously executed quality–revision or revision–quality prefixes across P1–P5, each already containing the date convention. Each continuation then acquires three relational updates in sequence, using the preceding stage's deployed program. The two prefix orders are paired with two continuation orders; this design evaluates continued acquisition under both sequences without isolating order as a causal factor. Reported total cost combines the recorded prefix acquisition with the three newly executed stages.

Table 7: Inherited application conditions preserve evidence-registration behavior while fitting the same teaching executions.

Event	Condition	Teaching	Apply	Audit	Evidence register
Alias	Inherited	10/10	25/25	25/25	25/25
Alias	Unconditional	10/10	25/25	25/25	15/25
Settlement	Inherited	10/10	25/25	25/25	25/25
Settlement	Unconditional	10/10	25/25	25/25	10/25
Approval	Inherited	10/10	25/25	25/25	25/25
Approval	Unconditional	10/10	25/25	25/25	20/25

Exploratory intervention: 15 paired parent/event programs differ only in whether the update inherits the existing applicability condition.

Table 8: Every executed continuation stage, including preserved prior successes and measured cost.

Sequence	Stage	New event	Before→after	Retained	Stage (s)	Cumulative (s)	With prefix (s)
P1/QR	1	Alias	11→15/15	67/67	11.35	11.35	467.70
P1/QR	2	Settlement	9→15/15	82/82	6.48	17.83	474.17
P1/QR	3	Approval	11→15/15	97/97	8.22	26.04	482.39
P1/RQ	1	Approval	11→15/15	67/67	12.66	12.66	467.23
P1/RQ	2	Settlement	9→15/15	82/82	6.52	19.19	473.76
P1/RQ	3	Alias	11→15/15	97/97	7.18	26.37	480.94
P2/QR	1	Alias	11→15/15	67/67	12.18	12.18	460.67
P2/QR	2	Settlement	9→15/15	82/82	7.86	20.05	468.53
P2/QR	3	Approval	11→15/15	97/97	9.37	29.41	477.90
P2/RQ	1	Approval	11→15/15	67/67	12.64	12.64	461.80
P2/RQ	2	Settlement	9→15/15	82/82	8.05	20.69	469.85
P2/RQ	3	Alias	11→15/15	97/97	8.83	29.52	478.68
P3/QR	1	Alias	11→15/15	67/67	11.21	11.21	306.83
P3/QR	2	Settlement	9→15/15	82/82	7.49	18.70	314.32
P3/QR	3	Approval	11→15/15	97/97	8.50	27.21	322.82
P3/RQ	1	Approval	11→15/15	67/67	12.53	12.53	307.87
P3/RQ	2	Settlement	9→15/15	82/82	6.80	19.33	314.67
P3/RQ	3	Alias	11→15/15	97/97	7.19	26.52	321.87
P4/QR	1	Alias	11→15/15	65/65	10.89	10.89	316.96
P4/QR	2	Settlement	9→15/15	80/80	7.96	18.85	324.92
P4/QR	3	Approval	11→15/15	95/95	9.34	28.19	334.26
P4/RQ	1	Approval	11→15/15	65/65	12.46	12.46	316.66
P4/RQ	2	Settlement	9→15/15	80/80	8.08	20.55	324.75
P4/RQ	3	Alias	11→15/15	95/95	8.63	29.18	333.38
P5/QR	1	Alias	11→15/15	67/67	11.35	11.35	225.26
P5/QR	2	Settlement	9→15/15	82/82	6.95	18.30	232.21
P5/QR	3	Approval	11→15/15	97/97	8.12	26.42	240.33
P5/RQ	1	Approval	11→15/15	67/67	13.17	13.17	226.81
P5/RQ	2	Settlement	9→15/15	82/82	6.94	20.11	233.75
P5/RQ	3	Alias	11→15/15	97/97	6.81	26.93	240.57

QR/RQ denotes quality then revision, or the reverse. Cumulative time covers the three new stages; with-prefix time adds recorded parent acquisition and the preceding two updates. P4 starts with 16/18 date successes. Retention counts all earlier successes at each stage. Prefix and continuation orders are paired.

D FEEDBACK BUDGETS

The feedback experiment crosses five parents, two events, targeted selection, and three random-query seeds. Budgets of zero, one, two, and four are prefixes of each query stream. The study uses sixty-five teacher executions. Programs with identical source are evaluated once on each identical task set, giving 225 physical evaluation episodes. Teacher execution is included in world-construction time. This experiment tests feedback separately from the successive updates above.

Table 9: Additional feedback: quality and cost at every query budget.

Event	Selection	Budget	Runs	Queries	Mean success	Added (s)	Update + query (s)	With parent (s)
Quality	Targeted	0	5	0.00	15.00/15	0.00	22.60	322.26
Quality	Targeted	1	5	1.00	15.00/15	16.45	39.06	338.72
Quality	Targeted	2	5	1.00	15.00/15	16.45	39.06	338.72
Quality	Targeted	4	5	1.00	15.00/15	16.45	39.06	338.72
Quality	Random	0	15	0.00	15.00/15	0.00	22.60	322.26
Quality	Random	1	15	1.00	15.00/15	16.41	39.01	338.68
Quality	Random	2	15	2.00	15.00/15	32.37	54.97	354.64
Quality	Random	4	15	2.33	15.00/15	37.55	60.15	359.81
Revision	Targeted	0	5	0.00	11.00/15	0.00	22.11	321.78
Revision	Targeted	1	5	1.00	15.00/15	27.89	50.01	349.67
Revision	Targeted	2	5	1.00	15.00/15	27.89	50.01	349.67
Revision	Targeted	4	5	1.00	15.00/15	27.89	50.01	349.67
Revision	Random	0	15	0.00	11.00/15	0.00	22.11	321.78
Revision	Random	1	15	1.00	13.67/15	23.92	46.03	345.69
Revision	Random	2	15	1.33	15.00/15	33.70	55.81	355.48
Revision	Random	4	15	1.33	15.00/15	33.70	55.81	355.48

Targeted selection uses five parents; random selection uses three draws per parent (15 runs). Queries and costs are means; budgets are prefixes of each run. Identical selected programs share evaluation. These runs are separate from the successive-update experiment.

E ADAPTING WORKFLOW AND SKILL METHODS

SkillCraft. The port retains save, get, list, and execute operations and nested skill execution. During the two teaching tasks, it returns execution-status and output-quality feedback. Static library loading uses the last execution status, treating absent history as success; task correctness does not filter the library. Every evaluation task starts from an independent copy of the acquired library, with ordinary online execution available.

The direct-execution control instead supplies the original public workflow entry and uses no additional model inference. The shared-function control exposes the same automatically discovered pure decision boundary as an editable child skill. Before acquisition, six paired checks—twelve native teaching executions—verify unchanged public products, task outcomes, and API counts.

Agent Workflow Memory. The port performs offline workflow induction with the original output budget and seeded workflow selection. It supplies all selected workflow memories and leaves the complete original examples available in the working environment. Both methods are adapted to the common business setting using the same deployed parent.

Shared execution budgets. Both online ports use seed 1729, an 80-turn actor budget, a 4,096-token cap per output, and a 55,000-token input limit. SkillCraft freezes its library after the two teaching tasks. AWM makes one 1,024-token workflow-induction request per event and supplies every workflow when the library contains at most ten. Direct-execution comparisons share the public entry binding and a 200-primitive-call task limit. Acquisition, online reasoning, resource execution, and interface-recovery costs are reported separately.

All fifteen Qwen3-8B SkillCraft revision runs terminate at the context limit before issuing a primitive AppWorld API call. Their 613 recorded model responses consume 28,299.55 seconds; fifteen

Table 10: Related-method execution on quality and revision updates. Acquisition model time and complete incremental time are separate columns; reuse counts cover 15 tasks.

Method	Execution	Event	Success	Model (s)	Update (s)	LLM calls	API calls
AWM 27B	Online	Quality	9/15	108.23	–	54	724
AWM 27B	Online	Revision	9/15	108.61	–	33	674
AWM 8B	Online	Quality	2/15	95.76	–	$\geq$471	3203
AWM 8B	Online	Revision	0/15	40.32	–	$\geq$691	1489
SkillCraft 27B	Online	Quality	6/15	441.79	–	$\geq$479	948
SkillCraft 27B	Online	Revision	4/15	251.50	–	$\geq$390	1262
SkillCraft 8B	Online	Quality	1/15	$\geq$3723.69	–	$\geq$452	32
SkillCraft 8B	Online	Revision	0/15	$\geq$1828.13	–	$\geq$613	0
SkillCraft 27B	Direct	Quality	9/15	441.79	–	0	603
SkillCraft 27B	Direct	Revision	9/15	251.50	–	0	626
SkillCraft 8B	Direct	Quality	9/15	$\geq$3723.69	–	0	603
SkillCraft 8B	Direct	Revision	9/15	$\geq$1828.13	–	0	626
SkillCraft 27B	Shared function	Quality	9/15	1982.50	–	0	603
SkillCraft 27B	Shared function	Revision	9/15	1286.83	–	0	626
Decision replay	Direct	Quality	15/15	0.00	21.06	0	604
Decision replay	Direct	Revision	15/15	0.00	21.24	0	625

27B: Qwen3.5-27B; 8B: Qwen3-8B. API counts exclude research inspection. $\geq$ marks recorded lower bounds with unmetered failed requests; – means unavailable or inapplicable. Direct execution uses the supplied entrypoint; the shared-function control exposes the decision routine for editing.

Table 11: Recorded model work during acquisition and online reuse. Reuse covers all fifteen evaluation tasks. Calls count completed model responses; unmetered failed requests during reuse are listed separately. $\geq$ denotes a time lower bound.

		Acquisition		Reuse		Unmetered
Method	Event	Calls	Model (s)	Calls	Model (s)	reuse requests
AWM 27B	Quality	1	108.23	54	1827.10	0
AWM 27B	Revision	1	108.61	33	964.13	0
AWM 8B	Quality	1	95.76	471	$\geq$16849.91	8
AWM 8B	Revision	1	40.32	691	$\geq$25548.90	13
SkillCraft 27B	Quality	19	441.79	479	$\geq$24335.00	2
SkillCraft 27B	Revision	17	251.50	390	$\geq$33343.13	6
SkillCraft 8B	Quality	52	$\geq$3723.69	452	$\geq$26843.82	14
SkillCraft 8B	Revision	21	$\geq$1828.13	613	$\geq$28299.55	15

additional failed requests have unmetered usage. The preceding two-task acquisition records 21 responses and 1,828.13 seconds, plus one failed request with unmetered usage.

The execution traces contain 482 rejected import statements, 48 unavailable module operations, one unsupported `with` statement, and 82 malformed tool-call JSON objects. No proposed code reaches the application executor, and no skill tool is invoked. The shared public-API interface states its host-access restrictions and available skills in the prompt; these outcomes reflect the model's invocation behavior under that interface.

On receipt quality, the completed Qwen3-8B online conditions score 1/15 for SkillCraft and 2/15 for AWM. SkillCraft's other fourteen runs terminate at the context limit. AWM has eight context-limit terminations and five completed runs with incorrect business outputs. AWM's revision condition scores 0/15, with thirteen context-limit terminations, one turn-limit termination, and one completed run with incorrect business outputs. Table 11 includes recorded inference work and unmetered requests.

SkillCraft 27B revision yields four successes, five incorrect business outputs, and six context-limit terminations. Both acquisition tasks produce incorrect business outputs. Acquisition and reuse consume 110 and 1,262 primitive API calls, respectively; model costs and unmetered requests appear in Table 11.

Its receipt-quality condition yields six successes, six incorrect business outputs, two context-limit terminations, and one turn-limit termination. Acquisition and reuse consume 168 and 948 primitive

API calls, respectively. The two failed reuse requests have unmetered usage; Table 11 reports all 479 completed responses and their measured cost.

F FROZEN NEW-INSTANCE CONFIRMATION

Before constructing this pool, we fixed all previously acquired programs: the parent and replay update for each relational event, three ordinary function-edit draws, and three thinking-selection draws. The ninety single-event views contain thirty underlying batches, each reused across processing, audit, and evidence registration. New identifiers, amounts, orderings, approved-correction combinations, mismatched evidence, missing context, and legacy metadata follow a fixed construction seed policy. Historical acquisition costs remain attached to these programs; this study charges new construction and native evaluation, not a second acquisition.

A further eighteen views combine relations with filtering and local/history state precedence. Six underlying batches each have three purposes. Two actual continued programs descend from the same original program, with paired quality/revision and relational-update orders. Their preceding programs are also fixed. The resulting 108 task identities share 36 underlying batches. A separate set of 24 construction-health tasks uses different identities and seeds. Reference executions validate construction labels through native APIs before method evaluation.

Table 12: Frozen programs on new relational instances. Each event contains ten underlying batches, each viewed through processing, audit, and evidence registration. Three original acquisition draws share these thirty tasks. Identical programs reuse their native evaluations.

Event	Method	Draw	Process	Audit	Evidence	All
Alias	Parent	–	6/10	6/10	10/10	22/30
Alias	Decision replay	–	10/10	10/10	10/10	30/30
Alias	Function edit	1729	10/10	10/10	6/10	26/30
Alias	Function edit	2718	10/10	10/10	6/10	26/30
Alias	Function edit	3141	10/10	10/10	6/10	26/30
Alias	Thinking choice	1729	10/10	10/10	10/10	30/30
Alias	Thinking choice	2718	10/10	10/10	10/10	30/30
Alias	Thinking choice	3141	10/10	10/10	10/10	30/30
Settlement	Parent	–	4/10	4/10	10/10	18/30
Settlement	Decision replay	–	10/10	10/10	10/10	30/30
Settlement	Function edit	1729	10/10	10/10	4/10	24/30
Settlement	Function edit	2718	10/10	10/10	4/10	24/30
Settlement	Function edit	3141	10/10	10/10	4/10	24/30
Settlement	Thinking choice	1729	4/10	4/10	10/10	18/30
Settlement	Thinking choice	2718	10/10	10/10	10/10	30/30
Settlement	Thinking choice	3141	10/10	10/10	10/10	30/30
Approval	Parent	–	6/10	6/10	10/10	22/30
Approval	Decision replay	–	10/10	10/10	10/10	30/30
Approval	Function edit	1729	10/10	10/10	8/10	28/30
Approval	Function edit	2718	6/10	6/10	10/10	22/30
Approval	Function edit	3141	6/10	6/10	10/10	22/30
Approval	Thinking choice	1729	10/10	10/10	10/10	30/30
Approval	Thinking choice	2718	10/10	10/10	10/10	30/30
Approval	Thinking choice	3141	10/10	10/10	10/10	30/30

Frozen replay updates solve all thirty views for each relation. The three function-edit draws score 78/90 for aliases and 72/90 each for settlement and approval; thinking selection scores 90/90, 78/90, and 90/90. These denominators count repeated program evaluations of the same tasks. The 28 logical conditions produce 792 task rows; exact program identity permits 462 physical scoring worlds across seventeen program evaluations. They record 22,924 native API calls and 661.83 cumulative group seconds. Task construction and reference checking use another 264 worlds, including health checks, with 4,943 reference APIs. Constructor APIs are not separately metered.

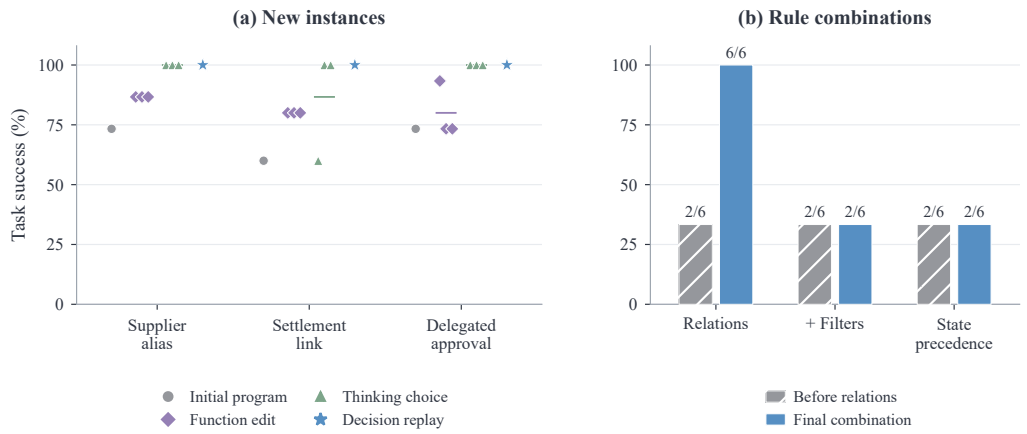

Figure 9: Frozen updates on new instances. (a) Each small mark is a prior acquisition evaluated on the same thirty views; offsets separate coincident marks, and horizontal segments give means. Gray marks are parents. (b) The two tested continued histories have identical per-task outcomes and are displayed once. Each category contains two batches in three uses; the final total is 10/18, versus 6/18 before the relations.

The mixed-rule failures identify two distinct interactions. A learned revision expression uses a correlated recording field instead of the convention's revision field. Separately, linked pending state overwrites a local settled entry before the existing return condition sees it. Both continued histories score 6/6 on mixed relations and 2/6 on each of filtering composition and state precedence. All evidence-register views remain correct. These fixed outcomes motivated the following development step and were not relabelled or reused as its new-instance confirmation.

G Maintenance scheduling protocols and complete results

The maintenance coordinator reads equipment windows, station bookings, capacities, and prior scheduled work, then creates reports and actual Todoist work orders or review tasks through AppWorld's public APIs. Windows may touch without overlapping; matching an identifier alone does not establish availability or capacity. Existing bookings and unrelated work must survive. The evidence use records the original directory interpretation without creating work orders.

Two authored parent programs implement this workflow with different decision structure and callee bindings. One uses positional arguments; the other uses reordered keywords and a Boolean use condition. Both qualify on normalized original-semantics teaching inputs. A directory identity change is taught by two ordinary correct executions. The same frozen relational grammar, condition substitution, and replay code are used without a scheduling-specific operator. The six evaluation scenarios cover new identities, interval boundaries, capacity, station conflicts, missing mapping, and legacy identifiers. Each appears in scheduling and evidence use, giving twelve views of 36 underlying requests. Each view contains six request decisions; counting both uses gives 72 decisions per parent.

A task succeeds only when its report and actual work items are correct, input and unrelated state are preserved, and no unexpected project items appear. The two parents and all model draws share this pool. Table 13 reports every condition.

Function editing and candidate selection both use Qwen3.5-27B thinking, seeds 1729 and 2718, a 55,000-token input limit and a 16,384-token reasoning-plus-answer limit. Function editing receives the complete parent and both correct public executions. Selection additionally receives all six unlabelled adapters, the unchanged-parent option, and captured arguments; it pays capture and construction. Function edits allow at most two format attempts; selection allows one request. All eight artifacts pass both public teaching cases with one generation and no format retry. Four edits nev-

Table 13: Maintenance scheduling across two provided program structures. Each uses the same six scenarios in two purposes. Both model controls use thinking and all eight artifacts pass public teaching. Native success checks reports, work orders, and unchanged state. Parents are provided; the unscoped ablation shares its selected update and has no separate acquisition time.

Parent	Method	Draw	Schedule	Evidence	All	Acquisition (s)
Positional	Parent	–	2/6	6/6	8/12	–
Positional	Decision replay	–	6/6	6/6	12/12	2.112
Positional	Without use condition	–	6/6	2/6	8/12	–
Keyword	Parent	–	2/6	6/6	8/12	–
Keyword	Decision replay	–	6/6	6/6	12/12	2.520
Keyword	Without use condition	–	6/6	2/6	8/12	–
Positional	Function edit	1729	6/6	2/6	8/12	756.379
Positional	Thinking choice	1729	6/6	6/6	12/12	697.512
Positional	Function edit	2718	6/6	2/6	8/12	320.236
Positional	Thinking choice	2718	6/6	6/6	12/12	256.006
Keyword	Function edit	1729	6/6	2/6	8/12	488.560
Keyword	Thinking choice	1729	6/6	6/6	12/12	704.078
Keyword	Function edit	2718	6/6	2/6	8/12	210.448
Keyword	Thinking choice	2718	6/6	6/6	12/12	355.874

ertheless change evidence behavior: they retain the original helper/guard but normalize identifiers afterward, outside that condition.

The local model uses bfloat16 weights and SDPA attention. Thinking decoding uses temperature 0.6, top-p 0.95 and top-k 20; plain decoding in the composition controls uses temperature 0.7, top-p 0.8 and top-k 20. Both use minimum-p 0 and repetition penalty 1.0. The plain Qwen3.5 profile adds an output-token presence penalty of 1.5; thinking uses zero. Model and decoding-source hashes accompany every request.

The shared teacher construction/execution takes 4.745 enclosing seconds and 18 setup plus 26 execution APIs. Provided-parent qualification is separately archived. Per-update times use enclosing intervals outside the native simulator context. The simulator freezes its internal clock, so in-world setup/execution/inspection timestamps are not used as elapsed-time measurements; phase API counts remain directly metered. Private native-state inspection uses no public API calls but is included in the enclosing interval.

The model service lasts 3,844.24 seconds across eight acquisitions. After admission to an empty GPU, other users start compute processes on the same device. Recorded inference times describe that shared-resource run; token, API and world counts are also retained. A command-transport error before model startup is separately archived. No native task or generation is restarted. Table 17 compares replay to warm native early stopping on the same candidate order and selected program.

H Composition follow-up: historical refinement and return conditions

This follow-up was designed after the frozen composition diagnosis. The original eighteen-view result remains intact. New namespaces, seeds, identifiers, and amounts separate teaching, construction-health checks, and confirmation. Two new correct operational executions cover revision-field ambiguity and opposing local/history state combinations: a local pending entry linked to settled history, and a local settled entry linked to pending history. They supply 24 captured decision constraints. Selection uses these public executions and the prior public teaching constraints.

For each continued history, the earlier revision stage has three training-consistent alternatives. We retain all three and cross them with absence/presence of the original-return prefix, yielding six candidates. The old argument expression is matched within the recorded caller/callee's first argument. All exact copies are replaced—one in the quality-first history and two in the revision-first history—while later enclosing transformations and other functions remain unchanged. The protected variant copies a supported leading prefix of parameter-dependent pure assignments and an initial condi-

tional return from the original program. Its expressions use the original inputs; the implementation does not name a particular status or business field.

Both teaching-only selections undergo native acceptance before new evaluation. The original confirmation has four new batches in each of three combinations and three purposes: 36 views, twelve underlying batches. Before learning started, a separately frozen extension added three batches where both interactions occur simultaneously, giving nine further views. The two sets retain separate denominators. Ablations remove one component from the same selected update. Neither set supplies feedback for revising this batch.

Table 14: Complete composition follow-up. The histories differ in their paired quality/revision and relational-update orders. Each uses the same 36 new views (12 batches); the simultaneous-interaction set has nine views (three batches). Retention is measured for the deployed combined updates and model controls. Dashes denote inapplicable or unreported cells, not zero outcomes.

History	Update	Draw	New	Joint	Old	Acquisition (s)
Quality first	Original combination	–	20/36	3/9	–	–
Quality first	Rule refinement only	–	28/36	3/9	–	–
Quality first	Return protection only	–	28/36	3/9	–	–
Quality first	Combined update	–	36/36	9/9	19/19	10.535
Revision first	Original combination	–	20/36	3/9	–	–
Revision first	Rule refinement only	–	28/36	3/9	–	–
Revision first	Return protection only	–	28/36	3/9	–	–
Revision first	Combined update	–	36/36	9/9	19/19	10.509
Quality first	Function edit	8171	20/36	3/9	19/19	121.824
Quality first	Thinking choice	8171	36/36	9/9	19/19	829.869
Quality first	Function edit	8179	20/36	3/9	19/19	88.997
Quality first	Thinking choice	8179	20/36	3/9	19/19	879.297
Revision first	Function edit	8171	20/36	3/9	19/19	106.524
Revision first	Thinking choice	8171	20/36	3/9	19/19	1446.176
Revision first	Function edit	8179	20/36	3/9	19/19	145.148
Revision first	Thinking choice	8179	20/36	3/9	19/19	1354.322

Strong controls and retention. Two seeds (8171 and 8179), two histories, and two methods give eight acquisitions. Ordinary function editing uses plain decoding and at most two format attempts; candidate choice uses thinking with one attempt. Both have a 16,384-token output limit and the same 55,000-token input limit. Controls receive the same current source, relevant historical source and edit records, old public constraints, and new correct executions. Choice additionally receives all six unlabelled options and captured arguments, paying that common prefix. Failed or publicly rejected proposals retain the supplied parent; confirmation outcomes never trigger repairs.

Table 15: All composition-control acquisitions, including rejected and length-limited outputs. Function edits use plain decoding; choice includes reasoning within the 16,384-token output cap. Accepted denotes public teaching acceptance. Incremental time includes client acquisition, reception, native acceptance, and the capture/construction prefix for choice.

History	Method	Draw	Accepted	Input	Output	Model (s)	Update (s)
Quality first	Function edit	8171	No	45144	1043	114.79	121.824
Quality first	Thinking choice	8171	Yes	52707	9520	815.44	829.869
Quality first	Function edit	8179	No	45144	761	82.18	88.997
Quality first	Thinking choice	8179	No	52707	9978	865.03	879.297
Revision first	Function edit	8171	No	44860	877	98.56	106.524
Revision first	Thinking choice	8171	No	52423	16384	1438.67	1446.176
Revision first	Function edit	8179	No	44860	1316	137.85	145.148
Revision first	Thinking choice	8179	No	52423	16384	1346.81	1354.322

One quality-first thinking selection passes teaching and reaches 36/36, 9/9 and 19/19. The four function proposals and the other quality-first choice fail public teaching acceptance. Both revision-first choices reach the 16,384-token length limit. All seven resulting deployments retain the prior program and score 20/36, 3/9 and 19/19. Original-task evaluation covers four unique program identities and 76 actual worlds; exact source, entrypoint and profile matches reuse outcomes for repeated deployments. The joint extension similarly reuses already recorded identical programs.

Input representation and complete costs. The initial representation exceeded the declared input limit before generation. It is retained as a zero-generation encoding failure, including incurred native fallback and service costs. Cross-block dictionaries, record columns and reversible historical-source edits then encode exactly the same public payload. Full decoding is checked for equality; the current parent source remains unchanged. Actual input lengths are 44,860–52,707 tokens. Models, seeds, public content, candidate order and budgets are unchanged.

Update times include the measured client acquisition interval, capture/construction where used, reception, native teaching acceptance and small recorded bookkeeping costs. Teacher construction/execution is 9.924 seconds for four worlds and 99 teaching APIs; it is a shared resource. Table 16 adds it once to the recorded historical path. Service startup/setup/teardown and unallocated service overhead are reported by group: the initial encoding attempt records 69.268 total service seconds, and the recovered run records 4,957.402, of which 4,900.931 is client acquisition. These group totals are not added again to already included call intervals. One-time encoding preparation was not independently timed; it remains unmeasured rather than zero. Native confirmation, constructor checks, archival transfer and manuscript preparation are research work, outside update acquisition.

Table 16: Recorded cost along the same quality-first history for a matched-quality comparison. Both methods reach 36/36 new views, 9/9 joint views, and 19/19 original tasks. The historical resource is identical. Model-service setup, encoding recovery, and research evaluation remain separately reported.

Cost component (s)	Combined update	Accepted choice
Recorded parent and earlier updates	482.389	482.389
New shared teaching construction/execution	9.924	9.924
Complete incremental update	10.535	829.869
Recorded path sum	502.848	1322.182

Table 17: Matched validation policies select the same final program. Both pay capture and construction. Counts include capture and all native candidate checks. API conventions differ across workflows: maintenance counts setup and execution; composition additionally records inspection and completion calls. Comparisons within a row share the same convention.

Workflow	Program/history	Replay (s)	Native (s)	Worlds	API calls
Maintenance	Positional	2.112	4.041	4 / 8	88 / 176
Maintenance	Keyword	2.520	4.380	4 / 8	88 / 176
Composition	Quality first	10.535	22.743	4 / 10	256 / 652
Composition	Revision first	10.509	22.637	4 / 10	256 / 652

The matched native control is a post-hoc validation-policy comparison on fixed candidates and teaching data. It does not read replay verdicts or confirmation outcomes. Both policies pay the same capture/construction prefix and select the same source. Enclosing update time is 10.535/10.509 seconds for replay and 22.743/22.637 for warm early stopping, approximately 53.6% lower. Native worlds fall from ten to four; recorded calls fall from 652 to 256. Warm-control module import takes another 1.044/1.055 seconds and is recorded separately.

I NATIVE INTERACTION SUPPLEMENT

The accompanying archive contains six unmodified original MP4s and two three-column comparisons: THOR draft, observation-decoder correction and combined components; Minecraft draft, state-access correction and ordinary full update. Full action traces, native final checks, per-action CSVs, source hashes and a key-action guide accompany the recordings. All original conditions are included, including unsuccessful programs and the successful full-update control.

Each panel starts at its own recording's first frame and keeps original playback speed. It switches to an explicit end card when that recording finishes. The initial world is shared within each com-

parison; action times are not synchronized by capture position. Minecraft's roughly ninety-second recording windows include waiting and a post-execution tail, while the three program executions last 10.83, 18.57 and 21.43 seconds. Action tables use each execution's own monotonic clock. THOR's 18/29/6 charged actions exclude one initial observation; Minecraft's 11/20/22 APIs and final inventory checks determine the outcomes. These known-case component recordings are separate from new-instance workflow confirmation.

J PROPOSAL ACQUISITION AND REPAIR

These studies compare how an initial local proposal, bounded repair, and a full-program fallback affect date-rule acquisition. The four policies are: full-program proposal with condition repair; local proposal followed by the same full fallback; local proposal with bounded condition repair before fallback; and local proposal with model-based candidate selection before fallback. Every policy returns the deployed parent if no candidate passes the public teaching executions.

Acquisition cohort	Policy	New rule	All tasks	Original	Seconds
Original five	Full + repair	16/40	66/90	95/95	432.3
Original five	Local + repair	30/40	80/90	95/95	404.7
Independent three	Full + repair	16/24	42/54	57/57	388.7
Independent three	Local + repair	16/24	42/54	57/57	592.1

Table 18: Date-update acquisition in the original and independent seed cohorts. Both procedures share the full-program proposal and bounded condition repair. Times include model attempts and teaching checks from the same supplied resources.

Table 19: Five paired acquisitions of one date-rule update. Each draw evaluates eight new-rule tasks, ten further evaluation tasks, and nineteen original tasks. All fallback procedures share the full-program proposal and bounded condition repair. Rejected updates retain the parent. Times include every used model request and teaching check.

Update policy	Accepted	New rule	All tasks	Original	Seconds
Full proposal + repair	2/5	16/40	66/90	95/95	432.30
Local $\rightarrow$ full	2/5	16/40	66/90	95/95	658.14
Local + repair $\rightarrow$ full	4/5	30/40	80/90	95/95	404.67
Local + choice $\rightarrow$ full	3/5	22/40	72/90	95/95	603.99

The replication uses new acquisition seeds on the same task identities. It retains every failed proposal and full-fallback cost. Small-model conditions ran on 80GB GPUs after serving failures on 24GB GPUs, with unchanged inputs, seeds, and model weights. The failed requests have unknown provider usage, reported separately from the completed runs.

The original cohort uses five predeclared acquisition seeds. Within each seed, the policies share one full-program proposal and its bounded condition repair; each policy that invokes this fallback is charged its complete cost. Acceptance requires equality with the supplied public teaching reports.

K CANDIDATE SELECTION, ARGUMENT BINDING, AND OUTPUT BUDGETS

Date-rule selectors receive the same parent and unlabelled candidate set, using either the full original material or the pure decision routine and captured public tests. Costs include capture and report alignment. Output caps include reasoning tokens. The receiver accepts an unambiguous candidate-index JSON object even when surrounded by prose; tables distinguish raw-format delivery from this recovered delivery. Missing executables are unscored. The 4096- and 8192-token conditions share the same three acquisition seeds.

Model / mode	Seed	New /8	Eval. /18	Original /19	Full	Local	Local + repair	Local + choice
27B plain	5511	8,8	18,18	19	412.9	535.0	535.1	535.0
27B plain	5527	8,8	14,14	19	377.2	607.6	607.6	607.6
27B plain	5539	0,0	10,10	19	376.0	633.5	633.5	633.5
27B thinking	5511	8,8	18,18	19	1129.6	2462.4	2462.4	2462.4
27B thinking	5527	8,8	18,18	19	1054.4	2399.2	2399.2	2399.2
27B thinking	5539	0,0	10,10	19	1315.7	2626.2	2626.2	2626.2
Qwen3-8B plain	5511	0,0	10,10	19	163.1	315.1	315.1	315.1
Qwen3-8B plain	5527	0,0	10,10	19	158.0	194.1	194.1	194.1
Qwen3-8B plain	5539	0,0	10,10	19	154.0	183.7	183.7	183.7
Qwen3-8B thinking	5511	0,0	10,10	19	180.0	264.3	264.3	264.3
Qwen3-8B thinking	5527	0,0	10,10	19	206.9	325.8	334.3	344.2
Qwen3-8B thinking	5539	0,0	10,10	19	195.9	378.0	412.4	395.5
Granite-3.3-8B plain	5511	0,0	10,10	19	191.6	390.3	390.3	390.3
Granite-3.3-8B plain	5527	0,0	10,10	19	350.6	557.1	557.1	557.1
Granite-3.3-8B plain	5539	0,0	10,10	19	233.5	270.1	270.1	270.1

Table 20: Fifteen paired date-update acquisitions. Comma-separated success counts give full-program and local-with-repair outcomes; the other two procedures have the same success counts in this cohort. The four time columns correspond to the procedures defined in Appendix J and include model requests and teaching checks. Small-model conditions use 80GB GPUs after failed 24GB serving attempts with unknown provider usage; hardware and runtime differences also affect cross-model times.

Table 21: Individual paired date-update acquisitions. Each seed supplies one full-program proposal, whose complete cost is charged to every procedure using it. Identical selected programs share evaluations.

Seed	Policy	New rule	Original	Model calls	Seconds
4231	Full proposal + repair	0/8	19/19	1	421.90
4231	Local → full	0/8	19/19	2	672.86
4231	Local + repair → full	0/8	19/19	2	672.91
4231	Local + choice → full	0/8	19/19	2	672.86
5387	Full proposal + repair	8/8	19/19	1	413.41
5387	Local → full	8/8	19/19	2	668.60
5387	Local + repair → full	8/8	19/19	2	668.60
5387	Local + choice → full	8/8	19/19	2	668.60
6421	Full proposal + repair	8/8	19/19	1	403.08
6421	Local → full	8/8	19/19	2	646.10
6421	Local + repair → full	8/8	19/19	1	252.45
6421	Local + choice → full	8/8	19/19	3	689.64
7541	Full proposal + repair	0/8	19/19	1	402.79
7541	Local → full	0/8	19/19	2	651.82
7541	Local + repair → full	6/8	19/19	1	259.93
7541	Local + choice → full	6/8	19/19	2	293.32
8663	Full proposal + repair	0/8	19/19	1	520.34
8663	Local → full	0/8	19/19	2	651.31
8663	Local + repair → full	8/8	19/19	1	169.46
8663	Local + choice → full	0/8	19/19	3	695.55

Table 22: Candidate selection from the same date-rule parent and public examples. Output budgets include reasoning tokens. Raw delivery requires the requested format; received delivery also accepts an unambiguous candidate index within surrounding prose. Evaluation scores executable proposals before acceptance; deployed fallback procedures appear separately. Time includes capture, alignment, model attempts, and native acceptance, and depends on hardware for cross-model comparisons.

Backbone / output cap	Raw format	Received	Accepted	Eval.	Seconds
Qwen3.5-27B, 256 tokens	0/3	0/3	0/3	–	38.46
Qwen3.5-27B, 1024 tokens	0/3	0/3	0/3	–	110.49
Qwen3.5-27B, 4096 tokens	3/3	3/3	3/3	54/54	253.38
Qwen3.5-27B, 8192 tokens	3/3	3/3	3/3	54/54	246.26
Qwen3-8B, thinking	5/5	5/5	1/5	64/90	164.49
Qwen3-8B, plain	5/5	5/5	0/5	50/90	19.30
Granite-3.3-8B, plain	0/5	4/5	0/5	40/72	47.45

Table 23: Repair procedures from one shared draft. Every procedure reaches the same task quality. Times measure model requests; native validation and other execution costs are reported separately.

Repair path	Model calls	Model seconds	Eval.	Original
Symbolic completion	1	439.22	18/18	19/19
Short LLM choice	2	491.84	18/18	19/19
Full feedback	2	866.81	18/18	19/19
Local feedback	2	880.55	18/18	19/19

Table 24: Argument passing and condition repair on one date-update draft. The added context argument must reach the decision routine for the learned condition to affect execution. All four interventions use the same draft and task set.

Intervention	New tasks	Original
Initial local edit	10/18	19/19
Condition search only	10/18	19/19
Argument binding only	12/18	19/19
Binding + condition repair	18/18	19/19

Validation strategy	Accepted stages	New tasks	Executions per stage	Seconds/stage
Local decision replay	20/20	15/15	4	22.0
Native, reused interpreter	20/20	15/15	64	110.4
Native, repeated interpreter	20/20	15/15	64	308.0
Native, reused + early reject	20/20	15/15	33.0	57.5

Table 25: Matched validation across five acquired parents, two update orders, and two stages: twenty updates. Every strategy selects the same program. Each event has fifteen evaluation tasks. Time includes preparation and required validation with a common learning interval. Interpreter startup is reported separately; Table 27 adds recorded parent acquisition.

Parent	Order/stage	New	Date	Original	Replay s	Cold s	Warm s	Early s
P1	Q-R/1	15/15	18/18	19/19	21.1	302.1	111.1	57.9
P1	Q-R/2	15/15	18/18	19/19	21.9	322.4	112.3	58.6
P1	R-Q/1	15/15	18/18	19/19	21.2	303.4	110.5	57.5
P1	R-Q/2	15/15	18/18	19/19	19.9	325.7	114.3	57.1
P2	Q-R/1	15/15	18/18	19/19	23.4	300.6	109.1	56.8
P2	Q-R/2	15/15	18/18	19/19	22.0	301.0	110.9	57.7
P2	R-Q/1	15/15	18/18	19/19	23.9	295.6	109.1	57.1
P2	R-Q/2	15/15	18/18	19/19	22.2	305.6	109.9	58.6
P3	Q-R/1	15/15	18/18	19/19	22.4	319.9	110.9	56.7
P3	Q-R/2	15/15	18/18	19/19	20.8	326.1	112.8	57.5
P3	R-Q/1	15/15	18/18	19/19	21.5	313.9	108.4	56.9
P3	R-Q/2	15/15	18/18	19/19	21.4	326.8	114.0	57.7
P4	Q-R/1	15/15	16/18	19/19	24.0	323.9	109.5	57.2
P4	Q-R/2	15/15	16/18	19/19	22.2	324.7	110.3	57.2
P4	R-Q/1	15/15	16/18	19/19	22.4	326.0	109.5	57.5
P4	R-Q/2	15/15	16/18	19/19	21.9	322.1	111.3	58.9
P5	Q-R/1	15/15	18/18	19/19	22.2	280.3	107.8	57.5
P5	Q-R/2	15/15	18/18	19/19	22.3	277.2	110.2	57.0
P5	R-Q/1	15/15	18/18	19/19	21.6	280.2	106.9	56.8
P5	R-Q/2	15/15	18/18	19/19	22.6	283.2	109.1	58.2

Table 26: Five acquired parents, two update orders, and two stages per order. Every validation strategy selects the same program. Times include preparation and required validation; interpreter startup is reported in Appendix R. P4 begins with 16/18 date successes, both of whose initial failures remain unchanged.

Parent	Order	Parent cost	Replay total	Warm total	Cold total	Early total
P1	Q-R	413.4	456.3	636.8	1037.9	529.9
P1	R-Q	413.4	454.6	638.2	1042.6	528.0
P2	Q-R	403.1	448.5	623.1	1004.7	517.5
P2	R-Q	403.1	449.2	622.1	1004.3	518.8
P3	Q-R	252.5	295.6	476.2	898.5	366.7
P3	R-Q	252.5	295.3	474.9	893.2	367.1
P4	Q-R	259.9	306.1	479.8	908.6	374.4
P4	R-Q	259.9	304.2	480.7	908.0	376.3
P5	Q-R	169.5	213.9	387.5	727.0	284.0
P5	R-Q	169.5	213.6	385.5	732.8	284.4

Table 27: Total recorded acquisition time in seconds: one parent acquisition plus two executed updates. Each parent uses its first accepted acquisition, including its generation failures and retries. Initial base programs and supplied demonstrations are given resources. Parent acquisition is charged from its record rather than repeated.

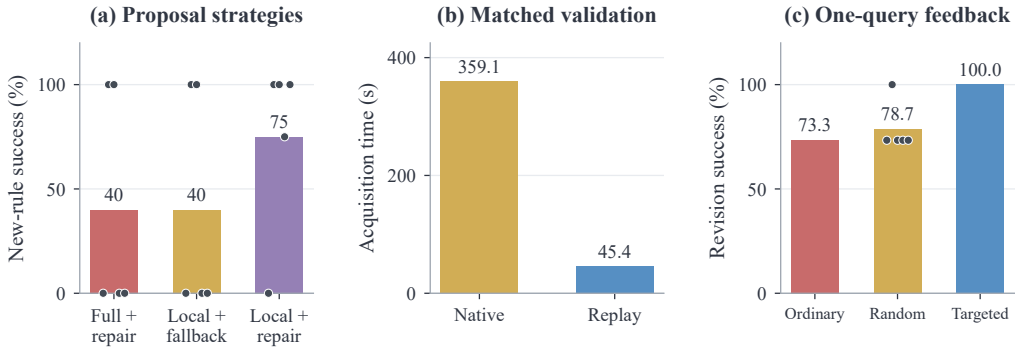

Figure 10: Proposal and query comparisons in the original cohort. A: date-rule success across five acquisition draws; bars show means and points show individual draws. Full-program proposals include bounded condition repair. B: revision acquisition with the same extra query and identical final program. C: revision success after one extra teaching execution, including all five random-query choices and the single deterministic targeted choice.

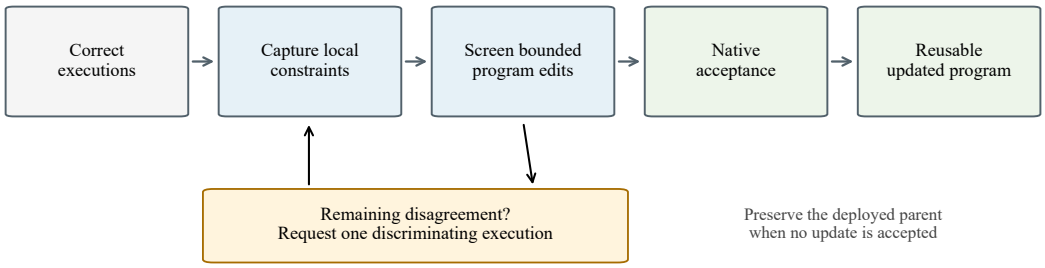

Figure 11: Program-decision update path. Pure replay filters candidates, an optional distinguishing execution supplies additional evidence, and native acceptance checks the selected update. Replay analyzes pure decision routines; ordinary proposals handle other program structures.

L VALIDATION COST, EVENT ORDER, AND REVISION OF EARLIER DECISIONS

The parent-diversity study uses all five distinct programs accepted on the original teaching tasks, originating from four acquisition draws; the unsuccessful fifth draw remains in the acquisition tables. Each stage applies the same finite grammar to its current parent. We compare the selected source and checked candidate outcomes against three native validation strategies: repeated interpreter creation, interpreter reuse, and reuse with rejection at the first incorrect public report. Retention is measured on exact previously successful task identities.

Table 28: Quality (q) and revision (r) updates in both orders from the date-aware parent. Every row also preserves 18/18 date tasks and 19/19 original tasks. Revision-rule ambiguity is evaluated separately under metadata shift.

Order	Stage	Update	Available-event success	Seconds	Teaching executions
qr	1	quality	quality: 15/15	21.66	4
qr	2	revision	quality: 15/15, revision: 15/15	20.19	4
rq	1	revision	revision: 15/15	20.38	4
rq	2	quality	revision: 15/15, quality: 15/15	21.25	4

The two replay sequences begin from the same date-aware parent. Model controls independently acquire each stage from their own preceding program. Matching exact program identities allows the paid transitions to compose eighteen additional program-state trajectories; these compositions reuse the recorded transitions.

Refining the earlier rule replaces its learned argument subtree while keeping the later quality wrapper. This tests whether a local revision remains compatible with a subsequent capability. The recorded edit-computation time excludes earlier learning, additional teaching, and acceptance; these costs are reported with their original acquisition paths.

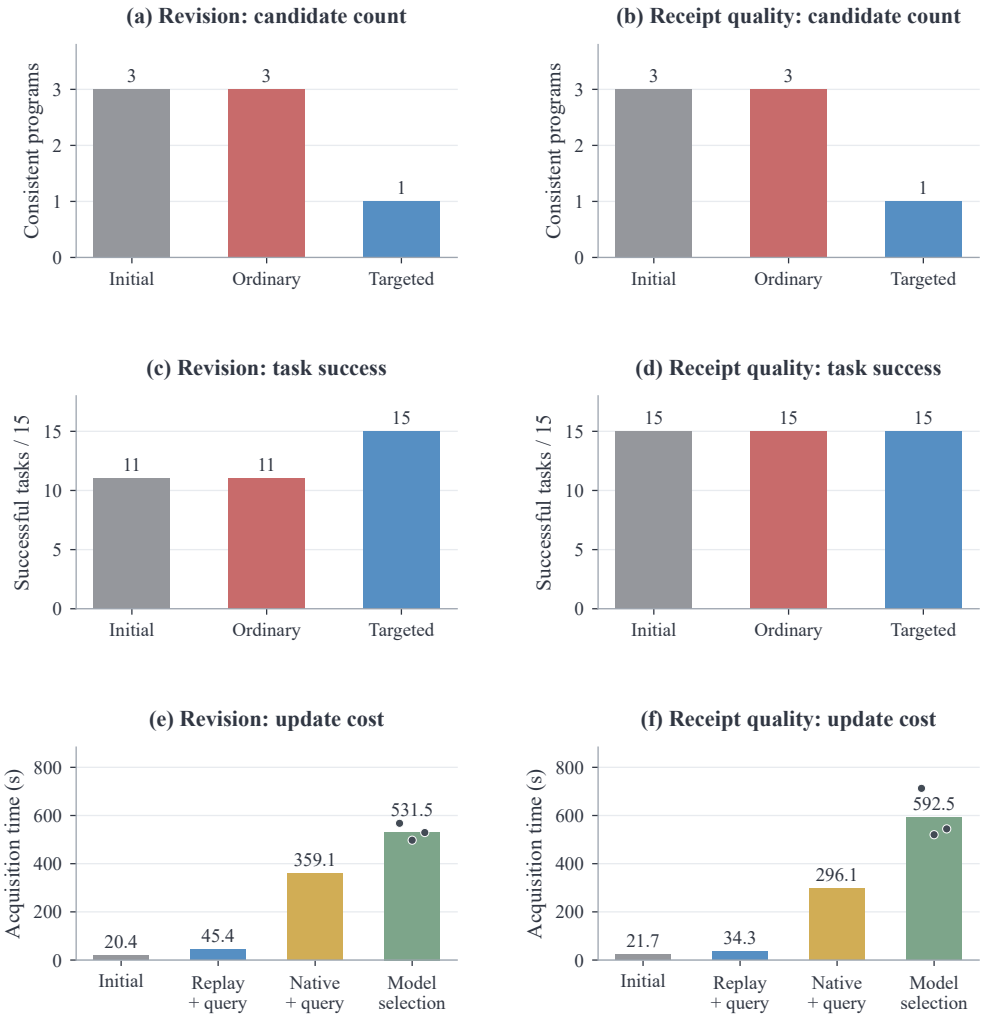

Figure 12: Revision and quality events expose different consequences of removing ambiguity. The initial revision rule fails under metadata shift; the initial quality rule already succeeds. Black markers show the three strong-model acquisition draws. Teacher resources differ from the strong selector, which uses no extra query. All scores refer to fixed development tasks.

M Additional teaching and random-query trials

Random queries sample uniformly from unique edits in the same typed public input space, independently of candidate disagreement. Duplicate edits arising in different per-claim tests receive one sampling entry. Each trial obtains one teacher execution and selects the first candidate satisfying

Table 29: Additional teaching on two business updates. Query and native-search procedures use the same extra teacher execution. Strong model selection uses the original two examples; its time averages three draws. Costs start from the deployed parent and original demonstrations. Identical final programs share evaluations.

Event	Method	Extra teacher	Shifted eval.	Seconds	Teaching executions
Revision	Two examples	0	11/15	20.38	4
Revision	+ ordinary example	1	11/15	29.98	5
Revision	+ disagreement example	1	15/15	45.37	8
Revision	Native search + same query	1	15/15	359.11	77
Revision	Strong model, two examples	0	15/15	531.50	2
Quality	Two examples	0	15/15	21.66	4
Quality	+ ordinary example	1	15/15	30.39	5
Quality	+ disagreement example	1	15/15	34.34	5
Quality	Native search + same query	1	15/15	296.08	65
Quality	Strong model, two examples	0	15/15	592.52	2

Table 30: Individual random-query trials with one teaching execution each. All trials use the same fifteen evaluation tasks per event. Candidate IDs index the fixed event-specific candidate order.

Event	Query seed	Remaining candidates	Candidate ID	Shifted eval.	Seconds
Revision	4231	2	31	11/15	33.77
Revision	5387	1	36	15/15	45.45
Revision	6421	2	33	11/15	44.13
Revision	7541	3	31	11/15	32.91
Revision	8663	3	31	11/15	33.04
Quality	4231	1	31	15/15	35.59
Quality	5387	3	31	15/15	34.69
Quality	6421	3	31	15/15	35.79
Quality	7541	3	31	15/15	34.99
Quality	8663	1	31	15/15	34.25

both earlier and added public constraints. The five random choices share task identities; an identical selected program reuses its recorded evaluation on the same task set.

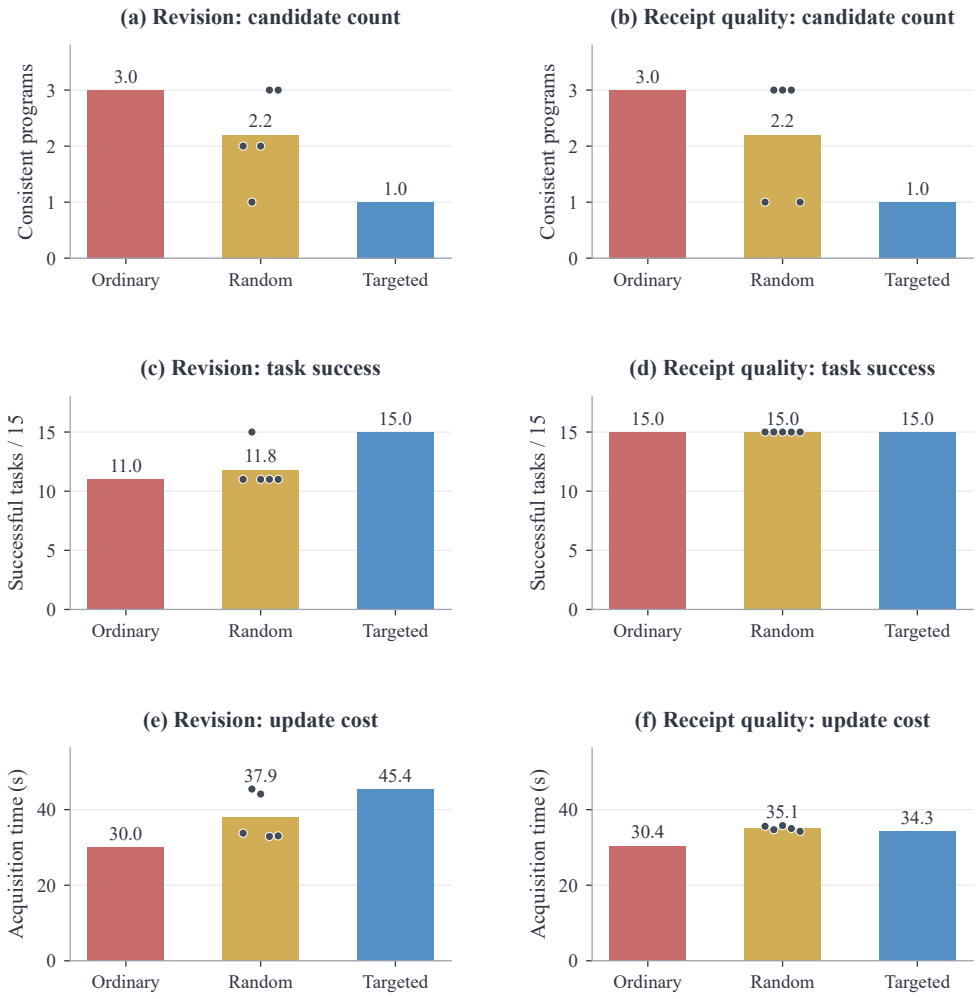

Figure 13: One additional teaching execution: individual random-query choices and their means. Querying reduces the consistent candidate set on both events, but improves success only for revision. Cost includes initial acquisition, query setup, teacher execution, capture, replay, and native acceptance for each compared path.

N MINECRAFT UPDATES AND VISIBLE EXECUTION

The resource-chain controller uses cardinal pathfinding with native physics. Navigation has no task-specific waypoints or teleportation, and mining and building require explicit program actions. All resource-chain programs share this controller and remain fixed during evaluation. The matched parent completes four of six original states and none of the four new layouts. Full updates retain all four previous successes in every draw; function updates retain three, zero, and four. The no-new-demonstration condition keeps the new request and all prior material, omitting only the new correct execution. Earlier controller attempts and their costs remain in the experiment archive.

Table 31: Resource-chain acquisition with one shared controller, four new layouts, and six original task states. Retention counts the four original states completed by the parent. Model time includes unsuccessful attempts; a missing executable is unscored. The parent has no measured update cost in this study.

Update	Seed	New /4	Original /6	Retained /4	Model sec.
Deployed parent	–	0/4	4/6	4/4	–
Full update	1729	3/4	4/6	4/4	318.75
Function update	1729	4/4	3/6	3/4	318.96
No new demonstration	1729	0/4	4/6	4/4	316.90
Full update	2718	0/4	4/6	4/4	636.63
Function update	2718	4/4	0/6	0/4	238.78
No new demonstration	2718	0/4	6/6	4/4	622.91
Full update	3141	4/4	4/6	4/4	741.44
Function update	3141	0/4	4/6	4/4	624.67
No new demonstration	3141	–	–	–	622.38

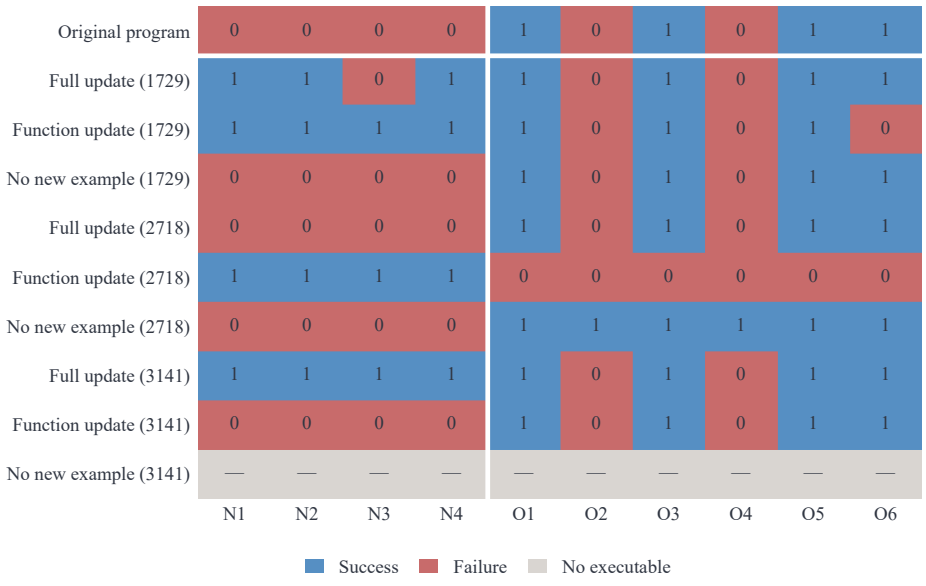

Figure 14: Resource-chain task outcomes under the shared controller. N1–N4 are new layouts and O1–O6 are original states. O2 and O4 are initial failures, so retention uses the other four original states. Dashes denote a missing executable.

The Minecraft videos use live Prismarine rendering driven by the server and bot. Policies act on structured public local state; the recorded frames visualize execution and are not policy inputs. The task table reports every evaluated resource-chain outcome.

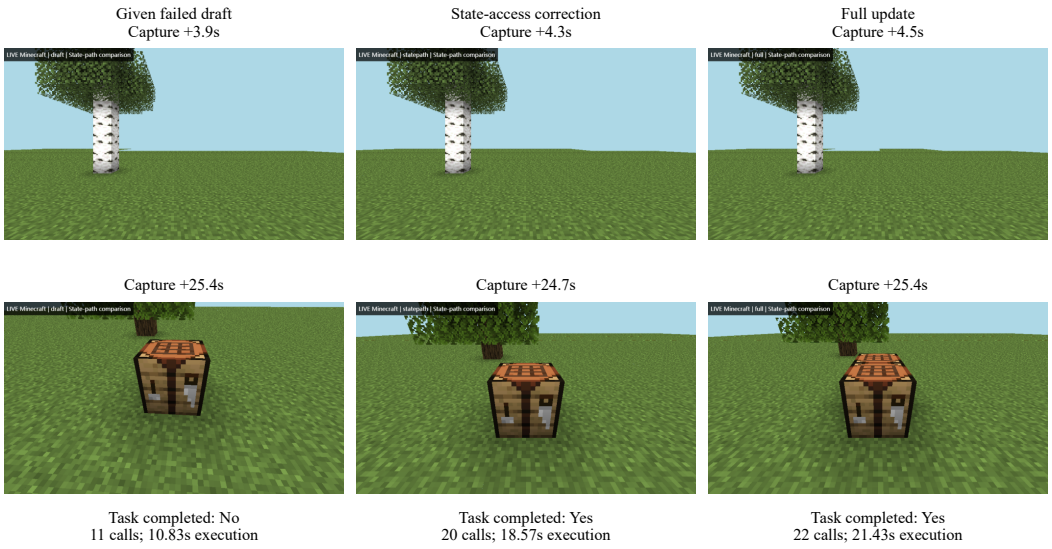

Figure 15: State-access repair in actual execution. The initial draft reads nearby blocks from the return wrapper; the repaired program accesses its observation member. All runs share the saved world, public request, and original controller. Unaltered frames are selected at fixed elapsed capture times; native records supply outcomes, API counts, and single-run execution times. Acquisition costs, including the shared failed draft, are reported separately.

A public-goal dispatcher preserves prior behavior. The new requests specify raw iron, a goal distinct from the original tasks. A dispatcher invokes the update for this publicly declared goal and the parent otherwise. Matching the source, world, request, and controller to previously recorded executions gives each of the eight delivered proposals the parent's four old successes and its own new-layout outcomes. These eighty outcomes are composed from the existing executions. Goal-based dispatch handles this disjoint-goal setting; the business updates instead require a condition within a shared decision.

O ALFWORLD EXECUTION AND EMBODIED UPDATES

Resource / control	Success	Regressions	Actions
Original program	60/60	0/60	642
Updated program, direct replacement	43/60	17/60	509
Public-interface dispatcher	60/60	0/60	642
Online, 12 demonstrations	59/60	–	778
Online, BM25 top 2	59/60	–	844

Table 32: Reuse and retention on sixty valid-unseen ALFWorld tasks, ten per family, held out until this evaluation. Regression counts lost successes of the original program. Program conditions receive an executable resource; online conditions receive demonstrations. All rows, including the public-interface dispatcher, are evaluated by execution.

The guide and demonstration controls use eighteen fixed TextWorld task identities and three execution seeds. They measure ordinary online execution with different forms of supplied experience. Query-selection cost is included where available; historical guide and demonstration acquisition is separate.

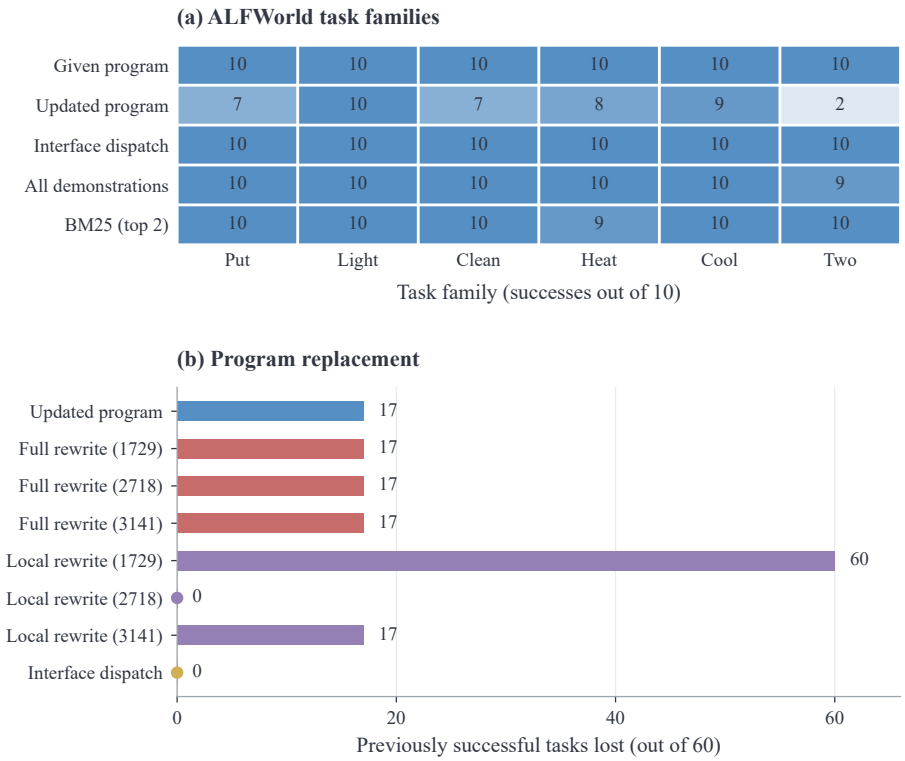

Figure 16: ALFWorld execution and program replacement. (a) Successes in six families of ten valid-unseen tasks. (b) Previously successful tasks lost after replacement; parentheses identify generation seeds. The interface dispatcher is directly executed. Online controls receive demonstrations; program conditions receive an existing executable.

Table 33: ALFWorld TextWorld controls on eighteen task identities and three execution seeds with Qwen3.5-27B. Input tokens include query selection where used; historical demonstration and guide acquisition are separate.

Condition	Success by seed /18	Total	Mean input /18
Direct actor	13 / 11 / 13	37/54	664,688
Bounded reflection	12 / 13 / 12	37/54	1,047,563
Short guide	16 / 15 / 16	47/54	514,033
Long guide	17 / 18 / 17	52/54	548,304
Fixed random 3 demonstrations	17 / 18 / 18	53/54	803,254
All 12 demonstrations	18 / 18 / 18	54/54	3,293,547
BM25, 1 demonstration	16 / 15 / 16	47/54	624,603
BM25, 2 demonstrations	16 / 16 / 16	48/54	1,215,771
LLM selection, 2 demonstrations	18 / 18 / 18	54/54	883,597

The NSI adaptation preserves upstream shared trajectories and skill-calling graphs. Original acquisition stops on inconsistent grounded facts. We test two adapters after converting duplicate facts and idempotent updates to Boolean set operations: one corrects a uniquely inconsistent observed fact using the public precondition of a subsequently successful teaching action; the other can also infer a unique atom read by that precondition. Both retain bounded model-feedback fallback. These are variants of the same NSI adaptation.

NSI acquisition outcome. After the fact-handling adaptations above, a bounded 27B recovery makes twenty-nine completed model calls totaling 5,467.83 request seconds and stops on a generated loop-entry label error. Together with preceding attempts, recorded model time is 34,989.06 seconds. Correcting an upstream variable-availability check leaves the generated graphs unchanged and still produces no valid joint library. Thus these standard-library acquisition attempts end without a reusable resource. The sixty-task standard experiment separately evaluates given-program execution and retention; it does not use a library from the failed induction.

Decode	Update	Received	Accepted	Model (s)	Acquisition (s)
Plain	Function	1/3	0/3	403.9	436.6
Plain	Full	3/3	0/3	664.8	828.6
Thinking	Function	3/3	0/3	571.6	687.7
Thinking	Full	2/3	0/3	786.3	870.1

Table 34: THOR current-view interface updates from two public successful executions. Acceptance requires both teaching rooms. Means include every failed generation, format retry, received proposal check, and native teaching setup. The shared original parent and demonstrations are given assets. Nine proposals are executable, but none passes both rooms; all deployed paths retain the parent (0/4 new rooms, 24/24 original TextWorld tasks).

The THOR study updates a TextWorld program to read a public current-view dictionary. Two successful public executions support acquisition under three seeds, full-program or function proposals, and plain or thinking decoding. All conditions share engine-derived visible-object names, oracle navigation, `forceAction` privileges, the action interface, and the execution budget. The receiver permits the already preloaded JSON module while charging all generation attempts. The unchanged parent solves twenty-four TextWorld tasks and none of the four new THOR rooms. Public-search references solve all four rooms; their source is withheld. Two thinking proposals complete one teaching room, but none completes both, so the parent remains deployed.

Component configuration	Teaching	New rooms	Old tasks
Decoder only	1/2	–	–
Decoder + goal-checked sequence + binding	2/2	4/4	16/24
Decoder + state binding	1/2	–	–
Decoder + open-loop sequence + binding	1/2	–	–
Decoder + goal-checked sequence	2/2	4/4	16/24

Table 35: Components applied to one given THOR draft. Evaluation covers four additional rooms from the official training split and twenty-four old TextWorld tasks. A dash denotes a proposal rejected on the two teaching rooms. Transformations use the public observations and draft without further model calls. The original parent scores 24/24 on the old tasks.

The component study uses the full thinking draft from seed 1729, whose acquisition and failed teaching checks cost 543.74 and 124.48 seconds. Its parser handles the initial observation prefix but misses ordinary JSON action returns, passing 2/56 supplied decoding examples. A three-candidate decoder grammar selects optional-prefix JSON decoding and passes 56/56. We then connect the longest observation–action sequence shared by the two demonstrations to the draft's target predicate, checking the goal after each action. This combination passes both teaching rooms and all four additional rooms. Omitting the goal check fails teaching acceptance; explicitly binding the navigation cache adds no quality benefit. All configurations modify the same given draft.

Direct substitution of the accepted programs yields 16/24 on old TextWorld tasks. A public-prefix dispatcher retains the parent for the old interface. Applying this rule to recorded inputs and exact

program identities composes 24/24 old and 4/4 new outcomes; the same control is available to every update policy. The later sixty-task standard study directly executes this dispatcher and obtains 60/60. Table 36 compares same-draft rewrites and selectors on twelve new-room tasks: eighteen policies require 96 physical executions because identical programs share evaluations.

Fresh automatic acquisition uses three new seeds, three backbones, and both proposal representations. The resulting fifty-four policy paths yield no accepted update. All retain the parent, obtaining 0/12 new-room successes and 24/24 old-task successes. The previously fixed component separately scores 7/12 in this second pool. A single display-initialization failure preceded world creation; the identical task and program resumed once, retaining the initial startup cost.

Same-draft method	Success /12	Seconds	Replication
Fixed component	7	759.4	one given draft
Candidate choice, plain	7,7,7	765.2	3 draws
Candidate choice, thinking	7,7,7	1168.7	3 draws
Ordinary local rewrite	6,0,6	1635.2	3 draws
Ordinary full rewrite	6,6,6	1302.8	3 draws

Table 36: Same-draft THOR comparisons on twelve new rooms. Times include the shared failed draft (543.74 model seconds), its failed teaching checks (124.48 seconds), additional model calls, and required native checks. Candidate selectors receive the automatically constructed options. Identical programs share evaluations. Startup, initial given resources, and research evaluation are reported separately.

Backbone	Delivered	Accepted	Outside grammar	Full s	Local + repair (s)
Qwen3.5-27B	5/6	0/6	5/5	1149.8	1801.2
Qwen3-8B	5/6	0/6	5/5	473.5	1014.8
Granite-3.3-8B	3/6	0/6	3/3	156.3	400.5

Table 37: Fresh automatic THOR acquisition with three paired seeds and two proposal formats per backbone. Thirteen proposals are executable; none passes both teaching rooms, and all lie outside the supported transformation grammar. Deployed procedures therefore retain the parent: 0/12 new rooms and 24/24 old tasks. The separately fixed component reaches 7/12 on the new-room pool. Costs include all generation and formatting attempts.

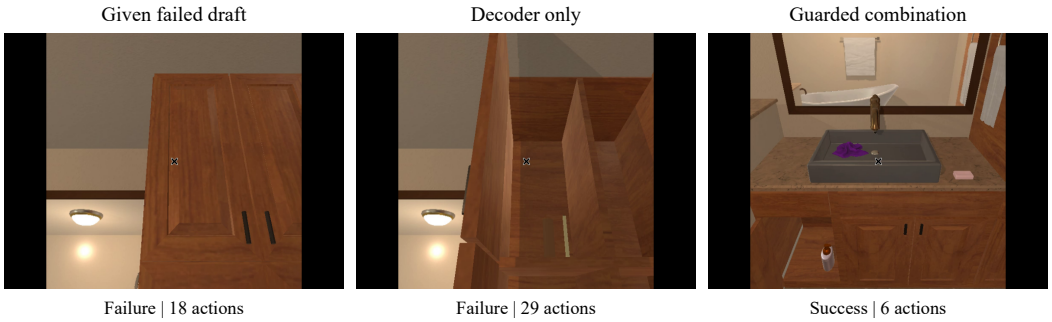

Figure 17: Actual Unity execution of one teaching task. The initial draft and decoder-only repair miss the target; the guarded action sequence finds and delivers it. Native goal checks establish outcomes. Unaltered Xvfb frames show each final state: success after six public actions, versus failure after eighteen and twenty-nine. These explanatory teaching-task repeats are separate from new-room evaluation. Policies use structured observations.

P ACQUISITION AND EXECUTION ACROSS STUDY CONDITIONS

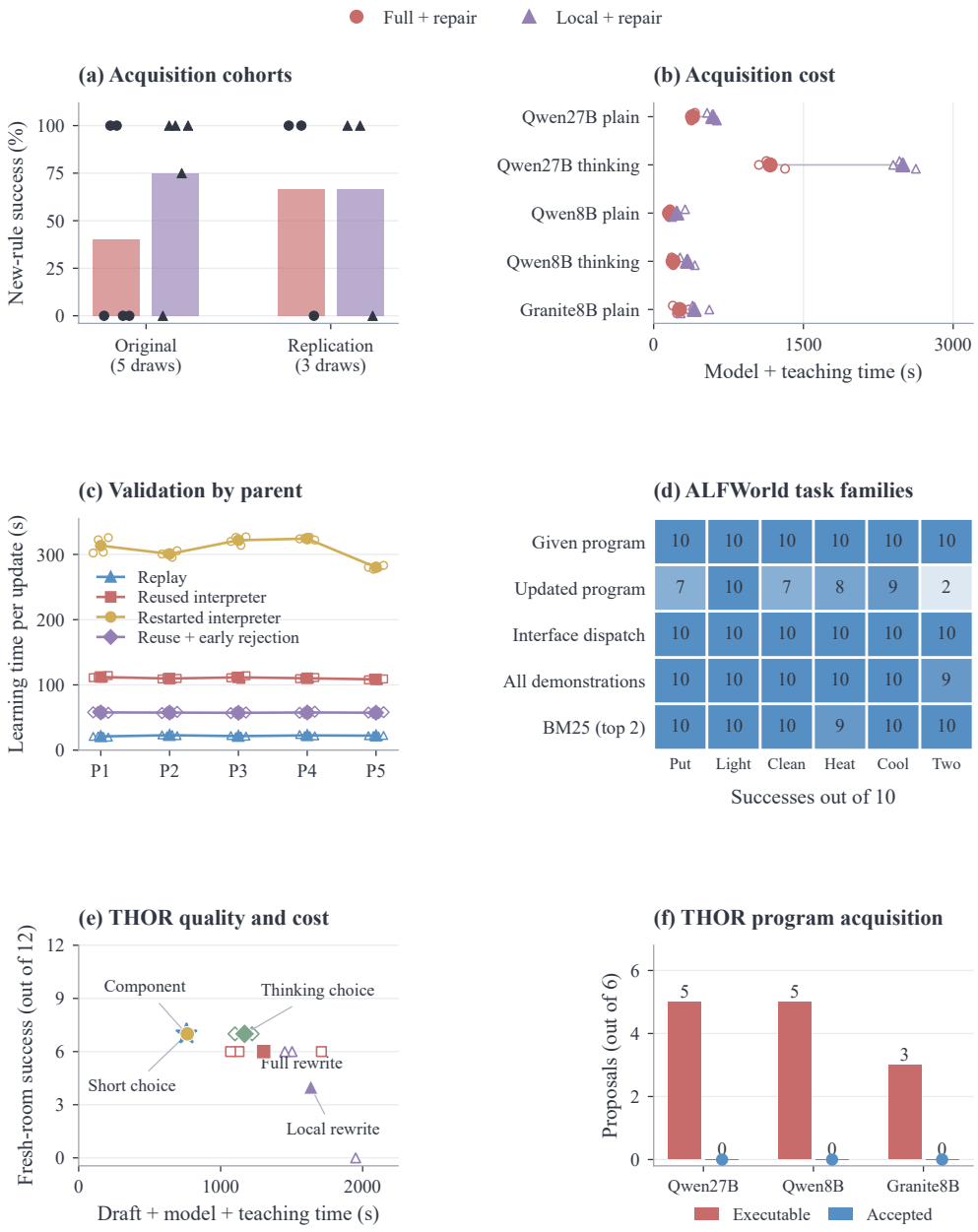

Figure 18: Acquisition and execution across conditions. (a) Original and replication date-update cohorts, with all acquisition draws. (b) Acquisition costs for full-program and local proposals with condition repair. (c) Four update stages per parent. (d) ALFWorld results with existing programs or demonstrations. (e) Same-draft THOR cost and new-room success; selectors share the component candidates. (f) Executable and accepted THOR proposals. In (b), (c), and (e), hollow marks show observations and filled marks their means; the component uses one given draft. Repeated stages share task identities.

Table 38: Fresh-instance evaluation of fixed programs. Date uses 36 task definitions; each other event uses 60. Denominators count these tasks across the listed acquisition draws. Preservation covers unchanged operational cases and evidence registration.

Event	Update procedure	Draws	New rule	Preservation	All
Date	Local → full	5	32/80	100/100	132/180
Date	Local + repair → full	5	60/80	100/100	160/180
Date	Strong 8192-token selector	3	48/48	60/60	108/108
Quality	Disagreement query	1	24/24	36/36	60/60
Quality	Two examples	1	24/24	36/36	60/60
Revision	Disagreement query	1	24/24	36/36	60/60
Revision	Ordinary extra example	1	14/24	36/36	50/60
Revision	Strong candidate selector	3	72/72	108/108	180/180

Table 39: All update procedures on fresh instances. The 156 task definitions share 32 underlying batches across consumer views and paired metadata conditions. Thirteen distinct event/program combinations require 612 candidate executions, reused across 3204 reported outcomes. Acquisition costs are reported with the original procedures.

Event	Update procedure	Draws	New rule	Preservation	All
Date	Full + repair	5	32/80	100/100	132/180
Date	Local → full	5	32/80	100/100	132/180
Date	Local + repair → full	5	60/80	100/100	160/180
Date	Local choice → full	5	44/80	100/100	144/180
Date	Parent adapter + replay	1	16/16	20/20	36/36
Date	Strong full-context selector	5	80/80	100/100	180/180
Date	Strong 8192-token selector	3	48/48	60/60	108/108
Quality	Deployed parent	1	0/24	36/36	36/60
Quality	Disagreement query	1	24/24	36/36	60/60
Quality	Ordinary extra example	1	24/24	36/36	60/60
Quality	Two examples	1	24/24	36/36	60/60
Quality	Uniform query	5	120/120	180/180	300/300
Quality	Full proposal	3	0/72	108/108	108/180
Quality	Function proposal	3	0/72	108/108	108/180
Quality	Strong candidate selector	3	72/72	108/108	180/180
Revision	Deployed parent	1	0/24	36/36	36/60
Revision	Disagreement query	1	24/24	36/36	60/60
Revision	Ordinary extra example	1	14/24	36/36	50/60
Revision	Two examples	1	14/24	36/36	50/60
Revision	Uniform query	5	80/120	180/180	260/300
Revision	Full proposal	3	0/72	108/108	108/180
Revision	Function proposal	3	0/72	108/108	108/180
Revision	Strong candidate selector	3	72/72	108/108	180/180

Q BUSINESS UPDATES ON FRESH INSTANCES

Programs, seeds, and task recipes were fixed before constructing the fresh instances. Date tasks span twelve batches and three consumer views; each other event spans ten batches, two metadata regimes, and three views. New instances preserve the established conventions and label rules while varying identifiers, amounts, cutoffs, thresholds, and non-admission metadata. All reference implementations pass native grading. Tables report every predefined proposal, random-query, strong-selector, and zero-query path without further program changes. Identical programs share evaluation outcomes, and confirmation execution is counted as research cost.

R IMPLEMENTATION AND COST ACCOUNTING

Program and task identity. Structured acquisition reports and native task records generate the numerical tables. The artifact manifest links each input and program to its digest. Teachers construct demonstrations through public APIs; their source and private task graders are excluded from learner inputs. Reusing a model response requires an identical request, model, and protocol. Reusing an evaluation requires an identical program, task, entrypoint, and controller. Each compared policy is charged every acquisition step it uses; shared physical execution is counted once in total research spend.

Measured intervals and missing costs. Reported acquisition includes failed generation, formatting attempts, candidate checks, and required native acceptance. Research-only output inspection is separated from business-execution API calls. Provider usage missing from an issued request remains unknown. Initial given programs, supplied demonstrations, model-serving startup, and research evaluation are reported separately from incremental learning.

Interpreter startup falls outside the matched learning intervals in Table 25. Warm-native processes average 133.76 seconds overall and 110.40 seconds within the learning interval; early-rejection processes average 68.16 and 57.52 seconds, respectively. Replay setup adds 0.13 seconds per two-stage sequence outside measured learning and research evaluation. All strategies use the same learning-time definition. Experiment output paths use unique identities, and corrected execution interfaces re-evaluate unchanged artifacts while retaining earlier failures and recovery costs.