CONTROLSCOPE: WORKFLOW REVISION AND RELIABILITY IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

How much of a running workflow should a language model agent revise? CONTROLSCOPE compares continuing generated code, editing the next tool call's data arguments, and replacing the unfinished workflow from the same public execution state. The nested permissions separate available repairs from the actions an agent selects. We evaluate one-time and repeated reviews across filesystem tasks, ALFWorld, and AppWorld. Across two source programs per task and three reasoning-reviewer draws on 20 filesystem tasks, FULL completes 15–16 tasks versus 13 for KEEP; across four fast draws it completes 10–13 versus 13. Fresh student-record confirmation reproduces a batch-read repair. ALFWorld fast panels yield KEEP/ARG/FULL scores of 85/86/87 on 87 tasks across 52 scenes and 134/134/127 on 134 tasks across four scenes; reasoning on the 87-task cohort also yields 85/86/87 with substantial review cost. An AppWorld V1 official-test panel of 585 task instances from 195 scenario templates shows small net differences. Frozen replays expose viable agent-written replacements interrupted by later revision in two failed file-organization runs. An offline source-trajectory midpoint comparison shows later reviews completing an insufficient repair. Five-call protection saves 19.4% of logged model output and loses one success across 20 fresh source runs. An argument-only shortcut shows that the broader sampled policy can overlook a cheaper successful edit available in both operation sets. These outcomes tie repair access to actual choices and subsequent execution.

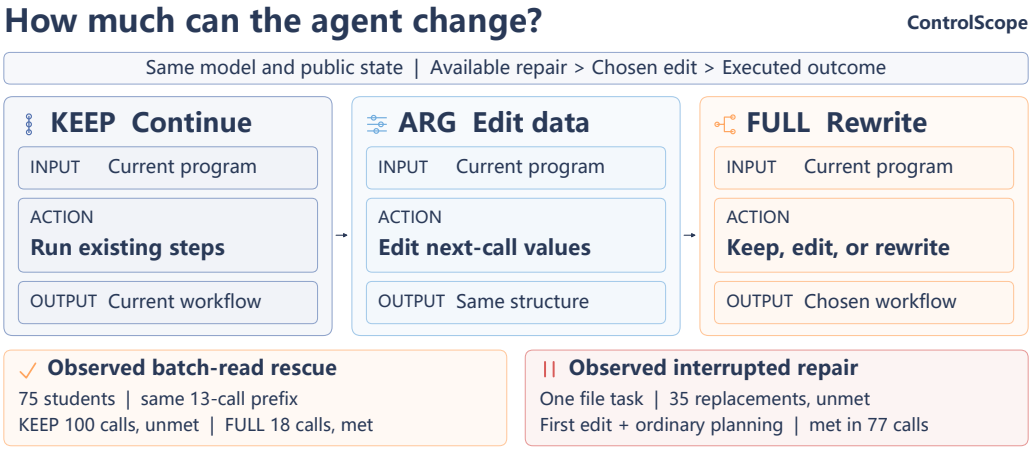

Figure 1. CONTROLSCOPE varies the part of an existing program an agent may revise from the same public state. FULL includes the smaller actions. The lower cards show a matched batch-read repair and a file-organization replay that retains its first replacement with ordinary planning.

1 INTRODUCTION

Language model agents increasingly act through executable workflows. A model may generate a loop over files, a sequence of application calls, or a plan for interacting with an environment. Program

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.

execution then carries out decisions that the model has already made. This division of labor supports efficient tool use, while making the authority to revise a running program an important architectural choice (Wang et al., 2024b; Kim et al., 2023; Qi et al., 2026).

A concrete example illustrates the choice. An agent must inspect student records and produce a contact list. Its generated program reads records individually. Under a fixed tool-call budget, a reviewer can correct the next file path or replace the loop with batched reads. Both reviewers receive the same public execution record and tool descriptions. Their permitted edits determine which repair they can make. Further workflow replacements can interrupt a useful continuation before it finishes its work.

We ask how much adaptation follows from expanding the scope of executable revision. CONTROLSCOPE compares three nested conditions, shown in Figure 1. KEEP continues the current program. ARG also allows edits to the next primitive tool call's data arguments. FULL additionally allows replacement of the remaining workflow. Here a primitive call invokes an original environment API directly, and a workflow is the program that organizes these calls. FULL can choose either smaller-scope operation at every opportunity.

Our experiments reveal useful adaptation and execution disruption. Across two source programs and three reasoning-reviewer draws, FULL solves two or three more tasks than KEEP out of 20; four fast draws range from three fewer successes to equal success. The AppWorld V1 official-test panel shows small aggregate differences across 585 task instances from 195 scenario templates. In file organization, frozen replays show that viable continuations can be interrupted by later revisions. At an early single revision boundary, 216 paired draws yield equal ARG and FULL terminal outcomes. Fixed review schedules expose a tradeoff between subsequent repair opportunities and model work. Fresh-subset confirmation and an argument-only shortcut connect available control to actual choices and workload.

We use three complementary diagnostic dimensions. *Repair availability* has a positive witness when a recorded permissible edit reaches the goal under an explicit continuation protocol. Candidate-only replay tests the edit through its block or call budget; retained-edit replay also allows ordinary planning. *Repair selection* records the chosen edit. *Execution stability* records later reviews and execution. Matched prefixes connect these observations to outcomes.

2 RELATED WORK

Agents that reason, plan, and execute. ReAct interleaves reasoning with environment actions (Yao et al., 2022), while Toolformer learns API use through language modeling (Schick et al., 2023). Plan-and-Solve structures reasoning into planning and execution stages (Wang et al., 2023). Code as Policies generates executable programs for embodied control (Liang et al., 2022), while ReWOO separates reasoning from external observations (Xu et al., 2023). Plan-and-Act separates a learned planner from an executor (Erdogan et al., 2025). CodeAct represents actions as executable Python (Wang et al., 2024b), and LLMCompiler schedules function calls through a compiler-inspired design (Kim et al., 2023). ReCode recursively decomposes code into actions at different granularities (Yu et al., 2025). LLM-as-Code places control flow in a program and uses the model within that structure (Qi et al., 2026). These approaches establish the architectural choices that motivate our comparison. We hold an already-generated program fixed and vary the changes a reviewer can make during its execution.

Feedback, repair, and revision. Reflexion uses verbal feedback across trials (Shinn et al., 2023), Self-Refine iteratively improves generated outputs (Madaan et al., 2023), and self-debugging uses program execution to support code correction (Chen et al., 2023). AdaPlanner refines plans through feedback within and across plans (Sun et al., 2023); Language Agent Tree Search combines reasoning, action, and search (Zhou et al., 2023). GraSP explicitly represents skill dependencies and compares typed local repair with global replanning (Xia et al., 2026). Its local-repair findings provide a direct precedent for scope-sensitive recovery. Anticipatory reflection explicitly addresses the tension between frequent plan changes and consistent execution (Wang et al., 2024a). Neuro-symbolic code validation grounds plans through symbolic checks and environment interaction (Ahn et al., 2025); plan–memory coupling addresses repeated failures in software repair (Zhang et al., 2026b). Our comparison measures nested action sets and the agent's actual choices from each set. Work on second-pass revision further shows that improvement depends on the information and structure

supplied by the initial draft (Ning et al., 2026a). Recent itinerary-revision experiments compare full replanning, hierarchical repair, and local editing while measuring feasibility and plan stability (Yuan et al., 2026). We replay selected operations from the same state under a shared model and backend, then observe how the revised continuation executes.

Intervention value and execution interfaces. Zhang et al. (2026a) evaluate interventions by branching from the same trajectory prefix and distinguish intervention value from continuation risk. DIAL learns when extra computation is beneficial from counterfactual exploration (Li et al., 2026). We use same-state branching to study the executable scope of a revision and the outcome of the operation an agent selects. Planning-horizon comparisons analyze how often agents return to the model (Otani et al., 2026). Model Context Protocol (MCP) design studies and enterprise tool-interface comparisons show that execution interfaces affect agent behavior (Felendler et al., 2026; Mak et al., 2026). Accordingly, our main comparisons share a tool backend, and our configuration analysis includes a matched-interface fast control. CaMeL separates control flow from untrusted data to enforce security properties (Debenedetti et al., 2025). We study task completion with system permissions held fixed, providing a complementary view of workflow control.

Interactive evaluation. AgentBench evaluates language models across interactive environments (Liu et al., 2023). ToolSandbox provides stateful tool-use evaluation (Lu et al., 2024). AppWorld supports interactive coding across application APIs (Trivedi et al., 2024), MCPMark supplies realistic tool tasks and executable checks (Wu et al., 2025), and ALFWorld connects language interaction with embodied household goals (Shridhar et al., 2020). Our measured endpoints span MCPMark-derived filesystem tasks, ALFWorld, and an AppWorld V1 official-test panel. We state each execution contract and the filesystem task-selection rules so that scope effects can be interpreted alongside the underlying benchmark contracts.

3 Measuring Revision Scope

3.1 A shared execution state and nested operations

Let h denote the complete public execution record available at a decision boundary. It includes the user goal, observed tool results, current program, pending call, and exposed program variables. Let s denote the corresponding environment state. A *continuation* is the code and queued tool calls that have already been generated and remain to execute. Branches begin from the same (h, s) and use the same underlying model, primitive APIs, and permissions.

We define the permitted operation sets by

$$\mathcal{A}_{\text{KEEP}}(h) = \{\text{keep}\}, \tag{1}$$

$$\mathcal{A}_{\text{ARG}}(h) = \mathcal{A}_{\text{KEEP}}(h) \cup \mathcal{P}(h), \tag{2}$$

$$\mathcal{A}_{\text{FULL}}(h) = \mathcal{A}_{\text{ARG}}(h) \cup \mathcal{R}(h), \tag{3}$$

where $\mathcal{P}(h)$ contains valid edits to the next primitive call's data arguments, and $\mathcal{R}(h)$ contains valid replacements of the remaining continuation. Argument revision preserves the pending API and subsequent program structure. Changing a path, identifier, or data value can still change later behavior through the program's existing branches. Workflow revision can change call selection, sequencing, loops, and dependencies.

The executor exposes explicit operations for keeping, patching arguments, and replacing the continuation. The same patch implementation serves ARG and FULL. Parameters that directly encode executable programs fall outside the data-argument comparison. Whole-reply FULL panels cover the current block and later tool calls queued in the same assistant reply. The two P1 fast panels use current-block replacement (Table 1).

Granted and realized scope. Granted scope is the set of operations available to the model. Realized scope is the operation it actually chooses. Inclusion of the smaller sets gives FULL access to every restricted choice. Consequently, any loss from the larger set reflects the deployed selection and execution policy, including its interaction with the interface and budget. We log both the granted condition and every realized operation.

We measure *reliability* as native terminal task success under a stated policy and budget. *Execution stability* records whether a selected revision persists through later review and planning.

3.2 SINGLE REVISIONS AND SUSTAINED POLICIES

A *local intervention* grants one revision opportunity at a fixed boundary. Subsequent scope decisions are KEEP. A *sustained revision policy* offers the assigned scope repeatedly during execution. These are separate experimental treatments because later revisions can change whether an earlier repair reaches completion.

Every condition retains ordinary planning when a code block or assistant reply finishes. In ALFWorld, every condition also retains the same ordinary failure-recovery procedure. This common planner can produce a new program after observing execution results. The scope comparison therefore measures control over an existing unfinished program within an otherwise functioning agent loop.

For binary terminal task success Y, we report paired differences

$$\Delta_{A,K} = \mathbb{E}[Y(\text{ARG}) - Y(\text{KEEP})], \qquad \Delta_{F,A} = \mathbb{E}[Y(\text{FULL}) - Y(\text{ARG})], \tag{4}$$

$$\Delta_{F,K} = \mathbb{E}[Y(\text{FULL}) - Y(\text{KEEP})]. \tag{5}$$

The paired execution unit is a task–source-program instance. Source programs and reviewer draws from the same task share its structure and remain grouped in interpretation. Filesystem uncertainty resamples the seven task categories.

3.3 REPLAY AND CANDIDATE EXECUTION

A *prefix replay* reconstructs all environment interactions preceding a selected boundary in an isolated workspace. We compare API names, arguments, observations, and terminal files with the recorded source. File-clock fields are handled through an explicit normalization that preserves task-relevant modification times. Hidden evaluation code stays outside the agent workspace. Each selected branch receives the same public record, while later observations evolve with its own actions.

We use two complementary replay diagnostics. First, an actual FULL argument patch is executed through both scope labels from the same state. This checks that the candidate also belongs to the lower-scope execution path. Second, an actual workflow replacement is retained while subsequent scope interventions are disabled. One endpoint evaluates goal completion by the replacement block or original budget. A separate endpoint evaluates complete agent recovery after the retained edit and common ordinary planning. We identify the endpoint for each reported witness.

4 EXPERIMENTAL SETUP

Tasks and source programs. An evaluation panel fixes a task set, one source program per task, and a reviewer configuration. A source program is sampled before the scope comparison. The main filesystem comparison contains 20 tasks across seven categories, selected from a 24-task MCPMark adaptation after inspecting instruction constraints. Four tasks restrict Python use and are excluded from this comparison; their scored outcomes remain in the full 24-task supplementary records. The adapter gives generated Python code access to the original filesystem APIs through a proxy. Each task has two independently generated source programs, reflecting prior evidence that implementation choice can change measured outcomes in automated research (Ning et al., 2026b). Native file-target verifiers determine task success under the stated execution contract.

ALFWorld tests household plans in two path-defined cohorts fixed before policy outcomes. All 87 valid_seen tasks outside development scenes cover 52 scenes; all 134 valid_unseen tasks cover four. Both cohorts use fast reviewers with required tool calls and an 8,192-token cap; malformed reviews become plan errors handled by ordinary recovery. The 87-task reasoning-reviewer panel uses automatic tool choice, a 16,384-token cap, and KEEP continuation after invalid reviews. Scores include common recovery.

An AppWorld V1 panel adds 585 official-test tasks across 195 scenario templates, paired by initial policy draw. It offers one review per natural block, with a 40-block task cap and 250 API requests per block. Its current-block replacement renews that block's allowance; the whole-reply filesystem panels share a fixed 100-call task cap. We report official-test outcomes as split-level aggregates.

Model and budget. Source generation and ordinary planning use served `deepseek-flash` in fast mode; reviewers use fast or high-effort reasoning mode. Matched configurations share native tool

schemas, an explicit tool-return instruction, automatic tool choice, and a 16,384-token per-request cap. Fast sampling uses temperature 0.7; reasoning uses the provider's reasoning-mode settings.

MCP conditions receive 100 primitive calls, 100 ordinary planning turns, and 1,800 execution seconds; review latency is recorded separately. ALFWorld uses 100 actions and up to five ordinary repairs. MCP and ALFWorld reasoning-reviewer errors fall back to KEEP; ALFWorld fast review errors enter ordinary recovery. An MCP ordinary-planner error ends the trajectory, with native grading of the existing files. Paired analyses use validated prefixes and worker execution.

Analysis panels. We distinguish the completed sustained-policy panels, the repeated local experiment, and task-family follow-ups. The local experiment freezes 36 eligible states from 19 tasks across two source programs. Each state receives three ARG and three FULL draws in each reviewer mode, yielding 432 decisions and 216 paired draws. Four task–program combinations lack a supported boundary. Each task–source pair reuses one cached KEEP baseline; sampled reviewed continuations share ordinary-planner responses only when their exact requests coincide.

The primary tables report counts so that small denominators remain visible. We retain positive, negative, and equal outcomes. A seven-category bootstrap supplies exploratory uncertainty for the MCP task set; the number of independent categories governs its precision. All task-family follow-ups are identified as generated variants or subsets, and repeated source programs retain their shared task identity.

5 RESULTS

5.1 WORKFLOW REVISION PROVIDES CONFIGURATION-DEPENDENT GAINS

Table 1. Sustained-policy panels on the same 20 tasks. P1/P2 are independent source programs; draws from one source reuse KEEP. Replace marks current Block or whole Reply. F W/L counts paired FULL wins/losses against KEEP.

Source	Review configuration	Replace	KEEP	ARG	FULL	F W/L
P1	Fast, required, draw 1	Block	13	13	12	0/1
P1	Fast, required, draw 2	Block	13	13	13	0/0
P1	Thinking, auto	Reply	13	13	15	2/0
P2	Fast, required	Reply	13	13	10	0/3
P2	Thinking, auto, draw 1	Reply	13	14	16	4/1
P2	Thinking, auto, draw 2	Reply	13	14	15	3/1
P2	Fast, matched auto	Reply	13	11	13	1/1

Table 1 reports two source programs and three reasoning-reviewer draws. FULL solves 15–16 tasks versus 13 for KEEP. P1 gains contact inference and English-student selection. Both P2 draws gain contact inference, file splitting, and grade scoring; draw 1 also gains English-student selection and loses file arrangement, while draw 2 loses legal solution tracing. Thus, the source program and reviewer draw shape which tasks benefit from revision.

The matched fast panel completes 13, 11, and 13 tasks under KEEP, ARG, and FULL. Its equal KEEP/FULL totals contain one gain in grade scoring and one loss in legal solution tracing. Relative to ARG, FULL gains grade scoring and a music-report task. The ARG/FULL reviewers record 29/16 rejected proposals, which remain in the scored policy outcomes.

The fast P2 required-tool configuration scores 13/13/10 for KEEP/ARG/FULL, while automatic-tool fast scores 13/11/13. P2 provides a whole-reply, automatic-tool comparison across fast and reasoning reviewers; P1 fast also changes replacement boundary. Forty P2 first requests match in messages, schemas, tool choice, token cap, and model identifier. Reasoning settings and stochastic draws differ. The seven-category bootstrap 95% intervals for P2 FULL-KEEP are $[-30.0, -4.2]$ and $[-14.3, 16.7]$ percentage points for required-tool and automatic-tool fast, and $[0, 38.9]$ and $[-10.0, 27.8]$ for the two reasoning draws. Full per-row intervals are in the supplement.

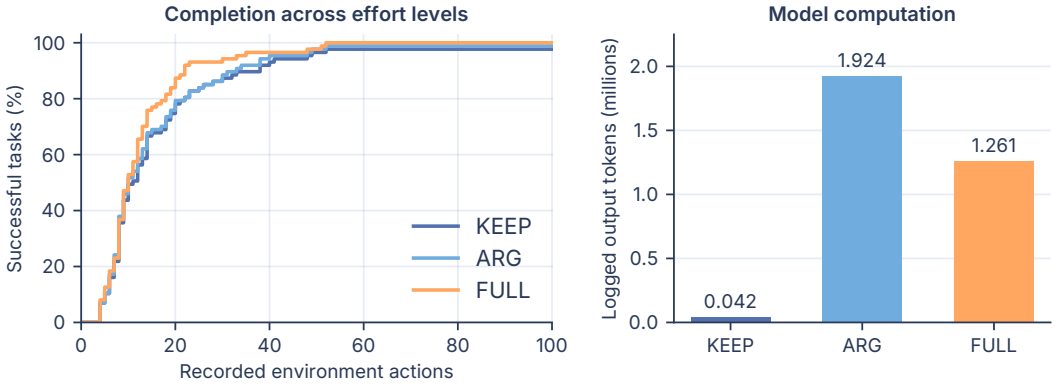

Figure 2. Recorded ALFWorld effort and model computation. The left panel gives the percentage of tasks completed within each environment-action threshold under the shared 100-action budget. The right panel reports logged output tokens, including reused requests. Both include common recovery.

5.2 Common recovery changes the visible benefit

On the 87-task reasoning-reviewer panel, initial-plan successes are 79, 81, and 85 for KEEP, ARG, and FULL. After ordinary recovery, these become 85, 86, and 87. Figure 2 shows execution effort. Within 20 actions, completion percentages correspond to 68, 69, and 76 tasks. FULL uses 1,124 actions versus 1,336 for KEEP and invokes ordinary repair twice versus 17 times. Logged input tokens total 0.221, 15.926, and 11.881 million; exact output totals are 42,481, 1,924,261, and 1,261,096 tokens. ARG/FULL record 51/16 invalid review replies, retained in task scores. FULL gains two terminal successes over KEEP with 29.7 times its output tokens. Its 34.5% lower total output than ARG concentrates 98.3% in two heating tasks with 28 invalid ARG reviews and none under FULL; FULL logs more output on 60 of 87 tasks. These configuration totals reflect reviewer validity and subsequent workload (Appendix C.3).

On the 87-task cohort, fast and reasoning reviewers both end 85/86/87. Under the same fast protocol, the 134-task valid_unseen cohort ends 134/134/127 across four scenes. The observed FULL direction changes across these fixed cohorts (Appendix C.3).

Within the 87-task reasoning-reviewer panel, two heating tasks account for all terminal differences. In scene 28, FULL replaces an unsuccessful stove-burner procedure with a microwave sequence after the same four initial actions and first recovery input. In scene 1, both ARG and FULL recover; ARG combines argument edits with the common planner. ARG records 15 and 13 invalid reviews in these two tasks, while FULL records none.

5.3 AppWorld official tests show small aggregate differences

Table 2. AppWorld V1 official-test aggregates. Task goal completion (TGC) counts successful tasks; scenario goal completion (SGC) is the percentage of templates with all three instances successful. K/A/F denote KEEP/ARG/FULL.

Split (tasks)	Templates	TGC (K/A/F)	SGC % (K/A/F)
Normal (168)	56	144/141/145	69.6/66.1/71.4
Challenge (417)	139	270/272/271	51.1/50.4/52.5

Across 1,755 completed AppWorld V1 runs, FULL exceeds KEEP by one task on each split (Table 2). Paired TGC gains are $+0.60$ and $+0.24$ percentage points; scenario-cluster bootstrap 95% intervals are $[-2.38, 3.57]$ and $[-2.40, 2.88]$. Paired FULL win/loss counts of 4/3 and 18/17 reveal changes in both directions. Each V1 FULL replacement also renews a 250-request block allowance. Its observed difference combines revision scope with this extra capacity, which favors FULL.

A separate V2 local diagnostic covers 86 development boundaries from 28 templates, with two draws and four permissions per boundary. KEEP, ARG, coordinated data editing (PARAMS), and FULL each succeed on 154/172 continuations; FULL selects 10 workflow replacements. V2 uses shared remaining-block quota and a private-runtime guard; FULL includes every PARAMS edit. In one pagination scenario, fixed KEEP, next-call edit, PARAMS, and workflow candidates succeed 0/3, 2/3, 3/3, and 3/3; online PARAMS and FULL selection succeed 0/3 and 1/3 (Appendix A.2).

5.4 A SINGLE EARLY INTERVENTION OFTEN LEAVES OUTCOMES UNCHANGED

All 216 paired local draws have identical ARG and FULL terminal outcomes. Two reasoning-enabled draws on the second file-splitting source improve over KEEP, and both scopes succeed in those draws. The other paired draws match the baseline outcome.

Realized choices help explain this pattern. Of 432 decisions, 391 ultimately continue unchanged, including 13 malformed responses and 12 service-error fallbacks. The remaining decisions contain 18 argument patches and 23 workflow replacements. The 12 service errors all arise from a single oversized context; excluding that state preserves the zero observed ARG–FULL difference. All 41 actual revision branches pass the saved-context, prefix, operation, and execution-budget checks.

The paired equality describes a fixed early boundary in 19 distinct tasks. Sustained policies reach additional states and can exercise different revision opportunities.

Visible runtime feedback. A paired information diagnostic gives both reviewers FULL authority once at the same natural execution boundary. A code-and-history view contains the generated program and prior public history; a feedback view adds current-block receipts, pending arguments, and runtime values. Across 17 eligible pairs covering 11 tasks, each view succeeds on 12/17, with 11 joint successes. Feedback selects nine replacements versus three, with one paired gain and one loss. This measures incremental visibility at the paused review (Appendix A.3).

6 HOW REVISION HELPS AND HURTS

6.1 AVAILABLE REPAIRS AND SELECTED EDITS

Batching reads. The English-student selection task requires combining basic student information with teacher recommendations. In one original source, a retained batch-read replacement reaches the goal after 94 total calls, whereas continuing the original loop reaches the 100-call budget without completing the task. Both branches share the first 88 calls. A second source yields the same mechanism at an earlier boundary, with the replacement completing the task at 28 total calls from an identical 26-call prefix. The resulting 19 selected students are independently checked against all original records. These frozen comparisons use recorded model choices and introduce no new review sampling.

Repairing a parser. Some student CSV records contain unquoted commas inside addresses. A workflow replacement can revise the parsing logic around reliable fields and produce correct results for all 150 students. The reasoning configuration succeeds repeatedly on the affected source, and the matched fast configuration also succeeds. The other natural source already parses the records correctly. This contrast ties the gain to a repairable property of the initial program. Some unsuccessful argument-review draws exhaust their output budget before returning a valid operation, and the recorded outcomes include those fallbacks.

Workload follow-up. Across 25-, 75-, and 150-student subsets with two seeds and natural source programs, both reviewer modes score 5/6, 5/6, and 6/6 for KEEP/ARG/FULL. The gain occurs on one 75-student seed-1 source. There, KEEP and ARG exhaust 100 calls; fast FULL switches to batch reads after an identical 32-call prefix and succeeds in 42 calls, while reasoning FULL succeeds in 10. Fast ARG selects three valid patches. Both 150-student sources already succeed under KEEP. The generated program's reading strategy shapes the observed revision opportunity.

Confirmation on new subsets. We select the discriminating 75-student setting after inspecting the workload exploration, then evaluate three new subsets with two independently generated source programs each. Both reviewer modes again yield 5/6, 5/6, and 6/6 successes (Table 3). On seed 101's first source, KEEP and ARG exhaust 100 calls; FULL succeeds in 18 fast-mode calls and 15

reasoning-mode calls. The fast repair follows 13 identical API calls and receipts, with zero rejected proposals or service errors. The second source already succeeds under KEEP on all three subsets. These comparisons preserve source-program dependence within the same task family.

Table 3. Fresh-subset confirmation on six task–program pairs per mode. Triplets follow KEEP/ARG/FULL. Calls and logged output include reused requests; output is shown in thousands to two decimals. KEEP is shared between modes.

Reviewer	Success K/A/F	Calls K/A/F	Output k K/A/F
Fast	5/5/6	183/191/105	18.47/31.09/23.80
Thinking	5/5/6	183/166/76	18.47/960.55/181.38

The confirmation also connects revision scope to computation. Relative to ARG, FULL uses 23.4% fewer logged output tokens in fast mode and 81.1% fewer in reasoning mode, computed from unrounded totals. A selected rewrite changes later tool workload, while reasoning ARG/FULL record 24/1 invalid-review fallbacks. Both contribute to the logged token contrast.

An argument-only efficiency shortcut. On seed 102's first source, the pending API supports batched input. The reasoning ARG policy expands its path list from 2 to 75 and succeeds in 11 calls. KEEP and the sampled FULL policy also succeed, each using 25 calls. The public state at the sixth-call boundary matches between ARG and FULL, which chooses to continue unchanged. Shared-patch replay succeeds under both labels in 11 calls with matching traces and files. The sampled FULL reviewer leaves this shortcut unused, adding 14 primitive calls while preserving success.

6.2 LATER REVIEWS SHAPE REPAIR COMPLETION

A successful candidate can lose its opportunity to finish when later reviews keep replacing it. We examine this behavior in the file-time classification task, where files must be moved into date-based directories and accompanied by metadata. The original execution encounters a genuine missing-parent-directory error. The agent's ensuing rewrite attempts address a real recovery need.

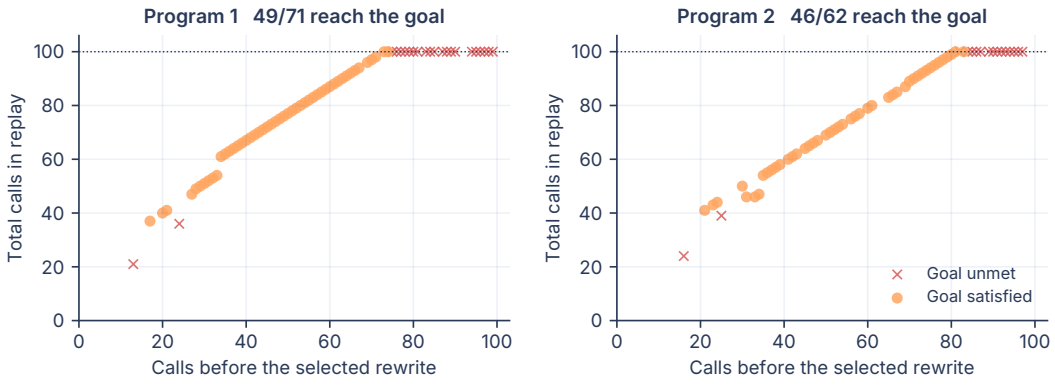

Figure 3. Frozen execution of every actual replacement in two failing runs of file-time classification. Program 1 uses P1 fast required draw 1; Program 2 uses P2 fast required. Each point retains the recorded prefix and executes the selected block to completion or the original budget. Goal satisfaction occurs for 49/71 and 46/62 candidates from one task with correlated prefixes.

For the first source, 49 of 71 actual replacements satisfy the task goal when retained through the block endpoint or the original call budget. For the second source, 46 of 62 do so (Figure 3). Forty-eight of the first source's successful candidates also finish their replacement block. These are correlated candidate-level replays within two runs of one task. They establish that useful continuations already existed within the failing policies' own outputs.

We also generate a small file-organization workload grid and select its discriminating setting for fresh data seeds. In three confirmation instances, KEEP and ARG succeed on all three, while FULL succeeds

on two. The failed run makes 35 replacements and reaches the 100-call budget. Retaining its first replacement and then allowing common ordinary planning succeeds in 77 calls. The first replacement alone ends after seven total calls with the goal still incomplete. This separates the value of preserving subsequent execution and planning from the ability of one code block to solve the entire task.

A broader retained-first-revision comparison covers every replacement-producing run in the P2 automatic-tool fast panel and first P2 reasoning draw. Sustained revision succeeds on 5/10 fast and 9/13 reasoning runs; retaining only the first revision yields 4/10 and 7/13, with zero paired gains and three losses. Later reviews complete repairs in those three changed outcomes. The interruption evidence remains case-level; this tested retention rule yields no panel-level gain.

Source-trajectory-defined midpoint schedule. The offline comparison uses P2 source programs and fixes each first-review boundary from the completed source trajectory length. This rule requires the baseline trajectory and serves as a paired diagnostic. One-time and continuous reviews share the first decision within each scope, and ordinary planning remains common.

Table 4 shows 12/20 successes under either ARG schedule, while FULL succeeds on 13/20 with one review and 14/20 with continuous review. The sole within-scope terminal change is English-student selection. After the same first FULL replacement, continuous review makes three later replacements and succeeds in 58 calls; one-time review exhausts 100 calls. The continuous FULL policy uses 294 reviews against 20 and logs 64.0k against 45.8k output tokens.

Table 4. Source-trajectory-defined midpoint schedule on 20 matched fast filesystem tasks. The shared KEEP baseline succeeds on 13/20. Edits count accepted patches and replacements; output includes reused requests. One-time legal ARG is scored after an ordinary-planner HTTP 400.

Scope	Schedule	Success	Reviews	Edits	Calls	Output (k)
ARG	Once	12/20	20	2	632	45.4
ARG	Continuous	12/20	313	12	615	59.0
FULL	Once	13/20	20	1	610	45.8
FULL	Continuous	14/20	294	17	596	64.0

Across 20 fresh source programs, KEEP/ARG/continuous-FULL/protected-FULL succeed on 13/12/14/13 tasks. Five-call protection activates on eight tasks, with zero paired wins, one loss, and 19 equal outcomes against continuous FULL. Reviews fall from 517 to 423 and logged output falls 19.4%. In the sole flip, continuous FULL selects a 300-path batch read and succeeds; protected FULL omits it and exhausts 100 calls (Appendix B.2).

7 IMPLICATIONS FOR AGENT DESIGN

Agent systems should expose both data edits and workflow replacement, since batch-read and parser repairs require wider control while a shared argument edit can save 14 calls. Review timing and valid-operation return rates belong in the same evaluation. Five-call protection saves 19.4% of output while losing one success.

8 CONCLUSION

CONTROLSCOPE compares KEEP, ARG, and FULL revisions from matched public states. Across two source programs per task and three reasoning-reviewer draws, FULL completes 15–16/20 tasks versus 13/20 for shared KEEP; four fast draws give 10–13/20 versus 13/20. Rewrites repair batching and parsing, with fresh student-record confirmation. A shared edit saves 14 calls, while frozen replays expose interruption in two runs of one file task. Both tested retention rules have zero paired gains. Retaining the first edit loses 3/23 replacement-producing runs, and five-call protection loses 1/20 tasks while saving 19.4% of output. ARG and FULL match in all 216 early local pairs; AppWorld V1 has small net differences with a quota favorable to FULL. Under a common fast ALFWorld protocol, FULL gains two tasks across 52 scenes and loses seven across four scenes. Edit scope, valid selection, and review timing jointly shape task success and model work.

AI USE STATEMENT

Generative AI assistants contributed to the research question, experimental design, code implementation, result analysis, literature review, manuscript writing, and figure production. Their outputs were checked against saved execution records, native task scores, and bibliographic sources. The authors take responsibility for the resulting claims and artifacts.

REFERENCES

Sanghyun Ahn, Wonje Choi, Junyong Lee, Jinwoo Park, and Honguk Woo. Towards Reliable Code-as-Policies: A Neuro-Symbolic Framework for Embodied Task Planning. arXiv preprint arXiv:2510.21302, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching Large Language Models to Self-Debug. arXiv preprint arXiv:2304.05128, 2023.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design. arXiv preprint arXiv:2503.18813, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks. arXiv preprint arXiv:2503.09572, 2025.

Yuval Felendler, Parth A. Gandhi, Idan Habler, Yuval Elovici, and Asaf Shabtai. From Tool Orchestration to Code Execution: A Study of MCP Design Choices. arXiv preprint arXiv:2602.15945, 2026.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM Compiler for Parallel Function Calling. arXiv preprint arXiv:2312.04511, 2023.

Ziming Li, Jiatan Huang, Xiaoguang Guo, Guilin Wang, and Chuxu Zhang. Same Signal, Opposite Meaning: Direction-Informed Adaptive Learning for LLM Agents. arXiv preprint arXiv:2605.06908, 2026.

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. arXiv preprint arXiv:2209.07753, 2022.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents. arXiv preprint arXiv:2308.03688, 2023.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. arXiv preprint arXiv:2408.04682, 2024.

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. arXiv preprint arXiv:2303.17651, 2023.

Hazel Mak, Susheel Suresh, Sahil Bhatnagar, Barry Wang, Chhaya Methani, and Alejandro Gutierrez Munoz. Is Bash All You Need? An Empirical Study of Tool Interfaces for Enterprise Digital Worker Agents. arXiv preprint arXiv:2609.11999, 2026.

Jingjie Ning, Xueqi Li, and Chengyu Yu. Revision or Re-Solving? Decomposing Second-Pass Gains in Multi-LLM Pipelines. arXiv preprint arXiv:2604.01029, 2026a.

Jingjie Ning, Shanshan Zhong, Xiaochuan Li, Ji Zeng, and Chenyan Xiong. One Run Is Not an Idea: The Implementation Lottery in Automated Research. arXiv preprint arXiv:2607.26587, 2026b.

Naoki Otani, Nikita Bhutani, Hannah Kim, Dan Zhang, and Estevam Hruschka. Do Agents Need to Plan Step-by-Step? Rethinking Planning Horizon in Data-Centric Tool Calling. arXiv preprint arXiv:2605.08477, 2026.

Junjia Qi, Zichuan Fu, Jingtong Gao, Wenlin Zhang, Hanyu Yan, Xian Wu, and Xiangyu Zhao. LLM-as-Code: Agentic Programming for Agent Harness. arXiv preprint arXiv:2606.15874, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv preprint arXiv:2302.04761, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv preprint arXiv:2303.11366, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. arXiv preprint arXiv:2010.03768, 2020.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive Planning from Feedback with Language Models. arXiv preprint arXiv:2305.16653, 2023.

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. arXiv preprint arXiv:2407.18901, 2024.

Haoyu Wang, Tao Li, Zhiwei Deng, Dan Roth, and Yang Li. Devil's Advocate: Anticipatory Reflection for LLM Agents. arXiv preprint arXiv:2405.16334, 2024a.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. arXiv preprint arXiv:2305.04091, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents. arXiv preprint arXiv:2402.01030, 2024b.

Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A Benchmark for Stress-Testing Realistic and Comprehensive MCP Use. arXiv preprint arXiv:2509.24002, 2025.

Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. GraSP: Graph-Structured Skill Compositions for LLM Agents. arXiv preprint arXiv:2604.17870, 2026.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. arXiv preprint arXiv:2305.18323, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint arXiv:2210.03629, 2022.

Zhaoyang Yu, Jiayi Zhang, Huixue Su, Yufan Zhao, Yifan Wu, Mingyi Deng, Jinyu Xiang, Yizhang Lin, Lingxiao Tang, Yuyu Luo, Bang Liu, and Chenglin Wu. ReCode: Unify Plan and Action for Universal Granularity Control. arXiv preprint arXiv:2510.23564, 2025.

Xiaofei Yuan, Yan Zhang, Shaobo Qiao, Huangleshuai He, Leyan Ni, Mingchen Ju, Lujia Yang, Sijia Xu, Yifu Tang, and Zhengyi Yang. Replan, Repair, or Edit? A Unified Empirical Evaluation of Travel Agents for Itinerary Revision under Resource Disruptions. arXiv preprint arXiv:2609.19654, 2026.

Chubin Zhang, Zhenglin Wan, Xingrui Yu, Jingxuan Wu, Qi Wen, Pengfei Zhou, Wangbo Zhao, and Ivor Tsang. Calibration Is Not Control: Why LLM-Agent Oversight Needs Intervention. arXiv preprint arXiv:2606.21399, 2026a.

Jiahao Zhang, Yifan Zhang, and Yu Huang. Coupling Planning with Episodic Memory in LLM Agents for Software Issue Resolution. arXiv preprint arXiv:2608.06811, 2026b.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. arXiv preprint arXiv:2310.04406, 2023.

A Execution Contract and Reproducibility

A.1 Scope boundaries and common planning

The filesystem agent emits Python blocks whose external effects occur exclusively through a proxy to the original Model Context Protocol (MCP) filesystem server. MCP is the protocol that exposes the tools and their input schemas. Python executes in an isolated worker with no direct task-filesystem mount or network access. The environment server receives a separate task workspace for each branch. Standard-library computation can transform returned data.

At each supported boundary, the worker exposes the pending primitive call, its arguments, JSON-serializable local and module variables, and printed output. The reviewer also sees the task history, current block, public tool schemas, earlier review decisions, and queued calls. A replacement cancels the pending call and substitutes new code for the unfinished assistant reply. Cancellation receipts preserve the tool-call protocol, and the ordinary model planner resumes afterward.

The two P1 fast runs permit current-block replacement. The P1 reasoning, P2, and local panels permit whole-reply replacement, including queued tool calls. A project-management source has a queued block after the reviewed block; the compared policies choose KEEP at that boundary and fail the task. Each panel is interpreted under its recorded replacement boundary.

A.2 AppWorld V1 protocol

The AppWorld V1 official-test comparison uses the same non-reasoning served model, public API descriptions, 16,384-token generation cap, and 40-code-block task cap across its three scopes. Its review occurs once per natural block. A FULL replacement edits the unfinished current block and starts a new 250-request block allowance; this extra capacity favors FULL in V1. The V2 local protocol shares the current block's remaining allowance, guards private-runtime access, and permits coordinated data edits. Its FULL action set includes KEEP, next-call editing, coordinated block editing, and workflow replacement. The filesystem protocol cancels queued calls in the same assistant reply and enforces a fixed 100-call task budget. Each panel follows its stated execution contract.

The official-test report contains 168 normal and 417 challenge tasks, with three task instances for each of 195 scenario templates. All 1,755 runs have final native scores and no outstanding infrastructure failure. SGC counts templates with three successful instances. Paired confidence intervals resample scenario templates. The frozen aggregate records task outcomes, API attempts, protected-criterion failures, review fallbacks, and quota rejections. Mechanism analysis uses development data and the official-test presentation uses the aggregate split-level results in Table 2.

A separate V2 local-intervention diagnostic covers 86 eligible natural development boundaries from 28 scenario templates, with two reviewer draws and four conditions for 688 completed continuations. KEEP, ARG, PARAMS, and FULL each succeed on 154/172 local runs. FULL selects 10 replacements that change API use while terminal success remains equal across conditions. V2 uses shared remaining-block allowance, a private-runtime guard, and coordinated data editing. These outcomes describe local continuations; V1 measures sustained full-task policies.

A fixed-candidate pagination diagnostic uses one AppWorld development scenario and its native evaluator. Across three repeated continuations, KEEP succeeds 0/3, a next-call page-limit edit 2/3, PARAMS edits to three data fields 3/3, and a fixed paging workflow 3/3. The candidates are frozen before evaluation. Separate online selection runs succeed 0/3 with PARAMS and 1/3 with FULL. This case witnesses available edits and incomplete selection in one app-API scenario.

A.3 Feedback visibility at a fixed boundary

For each of the 20 main filesystem tasks and two natural source programs, we freeze the first same-block boundary following an information-returning API call. The pending call must be a unique direct API site outside control-flow constructs. This source-based rule yields 17 eligible task–program pairs from 40 candidates, covering 11 distinct tasks. Only one candidate would qualify if the current block also had to be the task's first block. The selected pairs share environment state, execution prefix, remaining call budget, reviewer model, and a single FULL review opportunity. Later scope decisions are KEEP, while ordinary block-end planning stays available.

The code-and-history view (`CODE_ONLY`) contains the public history present when the current assistant reply was generated, its code and queued calls, API documentation, and the pending method and source line. The feedback view (`WITH_FEEDBACK`) adds current-reply tool messages, current-block receipts and printed output, the pending call's realized arguments, and public runtime variables. Both review requests use the same operation schema. Earlier blocks' observations can influence the common source program, while feedback-view module variables can carry earlier values. This comparison varies current-block runtime visibility while holding revision authority fixed.

All 17 pairs pass view-mask, exact-prefix, model-request, and native-execution checks, with no review or service fallback. Program 1 has eight eligible pairs, with code-and-history versus feedback success of 7/8 versus 6/8, replacement counts of 1 versus 4, and 215 versus 320 tool calls. Program 2 has nine pairs, with success of 5/9 versus 6/9, replacement counts of 2 versus 5, and 251 versus 324 calls. Each view succeeds on 12/17, with 11 joint successes, one feedback win, one loss, and 15 equal outcomes. Feedback selects nine replacements versus three and accumulates 644 versus 466 tool calls; two grade-scoring source units account for 167 of the 178 additional calls.

In program 1's grade-scoring task, code-and-history KEEP succeeds in 11 calls. Feedback selects a replacement whose CSV parser misreads unquoted address commas, and common planning exhausts the 100-call budget without producing the target files. In program 2's file-splitting task, feedback succeeds in 29 calls against a failed 33-call continuation under the code-and-history view. Its replacement repeats the pending file-information call, while subsequent ordinary planning writes the split files to the required directory. The two terminal flips thus involve the selected operation and the downstream continuation together.

A.4 RUNTIME AND INFORMATION CHECKS

The MCP source programs replay in independent workspaces. API arguments and non-clock observations must match before an intervention. File-creation and access times receive explicit replay handling; task-relevant original modification times remain preserved. When execution has already written a file, the corresponding output modification-time comparison is tracked separately. Raw clock differences remain available in the execution records.

Across 40 P2 matched-fast and reasoning first requests, messages, schemas, tool choice, token cap, and model identifier coincide. The P1 first-boundary queue is empty on all 20 tasks. The 174 ALFWorld policy branches have identical first public inputs to their corresponding baseline branches.

A malformed review response or review-stage provider error produces a recorded KEEP fallback, after which the existing program continues. A provider error in ordinary planning ends the trajectory, and the native verifier scores the files produced so far. Paired branches pass worker-transport, source-observation, and prefix checks. Fixed worker hash seeds reproduce source set displays; trajectory continuation preserves saved requests, responses, contexts, and decisions.

Fresh workload confirmation uses seeds 101–103, new source programs, and separately sampled review decisions.

A.5 BUDGETS AND MODEL REQUESTS

Exact-request caches couple ordinary planning draws within a paired panel where requests coincide. Review draws remain separately sampled. Reused KEEP trajectories and cached model responses are identified in resource accounting. Logical token totals describe the model work represented by the trajectories; successful uncached request totals describe a narrower component of actual API use. Monetary cost depends on provider pricing and cache treatment.

B STUDY COVERAGE AND OUTCOME ACCOUNTING

The completed main filesystem panels reuse the same 20 tasks. Additional reviewer draws and source programs provide repeated measurements on those tasks. The four instruction-incompatible tasks are pattern matching, structure analysis, structure mirroring, and duplicate student names. Their adapted outcomes remain in the full 24-task record. The seven-category grouping is desktop tasks, desktop templates, file context, folder structure, legal documents, papers, and the student database.

Table 5. Shared model settings and execution budgets.

Component	Setting
Model identifier	deepseek-flash
Fast reviewer	Thinking disabled, temperature 0.7
Reasoning reviewer	Thinking enabled, high reasoning effort
Matched reviewer interface	Automatic tool choice, 16,384 output tokens
MCP execution	100 primitive calls, 100 planning turns
MCP execution-time cap	1,800 seconds, review time accounted separately
ALFWorld execution	100 actions, initial plan and at most five repairs
Invalid review or review-stage provider error	Recorded continuation through KEEP

Code-repository tasks are kept outside the data-argument comparison because code-valued parameters can expand the effective revision set.

Table 6. Local revision decisions. Each row contains 108 draws across the 36 frozen states. KEEP counts include recorded fallbacks.

Reviewer	Scope	KEEP	Argument patch	Replacement
Fast	ARG	103	5	0
Fast	FULL	96	0	12
Thinking	ARG	98	10	0
Thinking	FULL	94	3	11

The local boundary rule chooses the first supported module-level internal boundary of the earliest multi-call writer block, with a read-block fallback. Function- and class-internal boundaries are excluded by this rule. All selected contexts have empty queued-call lists. Four task–program combinations are ineligible, leaving 19 distinct tasks represented by 36 states. Twelve context-limit failures belong to the same second-source paper-organization state. Excluding its draws preserves the observed equality of ARG and FULL outcomes.

B.1 SOURCE-TRAJECTORY-DEFINED MIDPOINT SCHEDULE

The matched fast schedule experiment uses all 20 tasks and the P2 natural source program for each. The completed source trajectory supplies its total N public API calls. First review then follows $\lfloor N/2 \rfloor$ calls; all 20 boundaries fall within the 100-call budget. The boundary is fixed before intervention runs, making this an offline paired diagnostic whose position requires the completed baseline trajectory. Both scopes share the public prefix, and each scope's one-time and continuous conditions replay the exact first review request, response, and decision. Continuous review can recur after a further public call; one-time review makes later scope decisions KEEP. Ordinary planning remains available. Five planner-stage HTTP 400 trajectories receive failing native scores; all 80 assigned conditions remain in the analysis.

The ARG conditions choose 2 and 12 patches under one-time and continuous review. The FULL conditions choose one replacement and 16 replacements plus one patch. In English-student selection, both ARG schedules and one-time FULL exhaust 100 calls. Continuous FULL makes four replacements, performs two batched reads of 150 paths each, and succeeds in 58 calls after ordinary planning writes the 19-student output. Both FULL schedules share the first replacement. This is the only within-scope terminal change. On legal solution tracing, continuous ARG changes agreement-version arguments and produces an incorrect CSV; both FULL schedules keep the source procedure and succeed. One-time ARG completes its scope review and then receives an HTTP 400 from ordinary planning after

36 public calls. The run ends without an output CSV. Its failure includes this planner-stage service event, while continuous ARG produces an incorrect CSV without a provider error.

Nine conditions record provider HTTP 400 responses. All four author-folder and four legacy-paper conditions have review-stage errors that fall back to KEEP. The legacy-paper runs still succeed; the author-folder runs also terminate after an ordinary-planning HTTP 400 and fail. One-time legal ARG has only a planner-stage HTTP 400. Three conditions exhaust the MCP call budget in English-student selection. All outcomes remain in the 20-task denominator for each assigned condition. Exact prefix checks cover the first review and continuation until continuous review gains its next opportunity.

B.2 FIVE-CALL EXECUTION PROTECTION

A second prospective panel generates one new natural source program for each of the same 20 instruction-compatible filesystem tasks before running any policy condition. It compares KEEP, ARG, continuous FULL, and FULL with five completed public calls protected after each selected workflow replacement. This rule uses observed call counts during execution. Ordinary block-end planning remains available. All conditions share the 100-call and 100-planning-turn budgets. The first selected replacement and every earlier review decision match exactly between the two FULL schedules. The scheduler pilot is excluded from the formal panel. All 80 assigned policy runs reach native scores without a technical failure.

The respective KEEP/ARG/continuous-FULL/protected-FULL success counts are 13/12/14/13. Protection activates on eight tasks and yields zero paired wins, one loss, and 19 equal outcomes. Continuous and protected FULL respectively use 537/551 primitive calls, 517/423 scope reviews, 40/30 replacements, 68,025,157/46,245,343 logged input tokens, and 79,075/63,731 logged output tokens. Logical token totals include replayed responses. English-student selection supplies the only terminal change. Both FULL schedules share a first replacement after seven calls. Continuous FULL selects a 300-path `read_multiple_files` call at call 88 and writes the target answer at call 89, succeeding in 90 calls. Protected FULL selects no batch read and reaches the 100-call budget. Provider and budget failures remain in the assigned denominator.

The workload confirmation fixes 75 students across new subset seeds 101–103 and two new natural programs each. These six task–program pairs share the original 150-record pool. All six source scores, 24 scope conditions, and first public inputs pass validation. The confirmation runs use separate source programs and reviewer decisions. Both exploration and confirmation retain every outcome direction.

The file-organization exploration spans eight generated instances and yields 8/8, 8/8, and 7/8 successes. Three fresh seeds at 16 files and three directory levels yield 3/3, 3/3, and 2/3. Both sets remain grouped within their respective task families.

C MECHANISM DIAGNOSTICS

C.1 SHARED ARGUMENT OPERATIONS

Every effective FULL parameter candidate in the audited main panels is replayed under ARG and FULL labels from its recorded prefix. The selected candidate can follow earlier workflow replacements. Its paired execution checks isolate the shared operation implementation. They preserve the distinction between a candidate reachable from the recorded FULL history and an independently sampled ARG policy. The three FULL parameter candidates in the local-repeat experiment also yield matching API traces, file bytes, and scores under both labels.

The confirmation replays the actual ARG patch from seed 102's first source under both labels at the sixth-call boundary. Later scope decisions are KEEP and ordinary planning remains available. Both replays succeed in 11 calls with matching traces and files. The sampled KEEP, ARG, and FULL policies all succeed; the patch reduces calls from 25 to 11 and witnesses an efficiency selection gap.

Confirmation invalid-review fallbacks are 2/2 for fast ARG/FULL and 24/1 for reasoning. All remain in the scored outcomes; service-error fallbacks are zero.

C.2 PRESERVING A SELECTED REPLACEMENT

The candidate-level file-organization replay in Figure 3 freezes each actual revision and executes its replacement block within the remaining original budget. Goal satisfaction is checked separately from normal block termination. This distinction accounts for the first source's one successful candidate that reaches the goal at the budget without finishing the block.

A commitment diagnostic retains the first actual replacement and common ordinary planning while disabling later scope changes. All three generated confirmation instances succeed in that condition. The failed sustained-policy instance is rescued in 77 calls; its replacement block alone stops after seven calls with an incomplete goal. Later ordinary planning completes the repair.

The first-revision comparison includes all 10 fast automatic-tool P2 runs and 13 runs from the first P2 reasoning draw that replace a workflow. All 23 prefixes pass replay checks; later scope decisions are KEEP while ordinary planning remains available. Fast success changes from 5/10 to 4/10, and reasoning success from 9/13 to 7/13, with zero paired gains and three losses. The losses concern grade scoring in both modes and English-student selection in reasoning mode. Retained author-folder runs end at the model context limit. Exact-request caching couples ordinary planning where requests coincide. Capacity terminations, equal outcomes, and losses remain scored.

C.3 ALFWORLD RECOVERY AND COMPUTATION

Both ALFWorld cohorts were frozen from official paths before policy outcomes. The 87-task valid_seen cohort excludes 24 development scenes and covers 52 scenes; all 134 valid_unseen tasks cover four. Under the same fast protocol, KEEP/ARG/FULL score 85/86/87 and 134/134/127 respectively. The 87-task reasoning-reviewer configuration also scores 85/86/87 on the same source plans and budgets. It uses automatic tool choice, a 16,384-token review cap, and KEEP continuation after invalid reviews. We detail this panel for its 52-scene coverage and complete review-outcome accounting. The fast 134-task cohort supplies the opposite direction.

The 87-task reasoning-reviewer panel has 26 effective ARG patches, 31 FULL replacements, and one FULL argument patch. The shared patch passes action-trace and native-score replay. Invalid reviews total 51 for ARG and 16 for FULL; FULL also has one service-error fallback. All 261 scored trajectories meet the action and repair budgets. Scene 28 has 15 ARG fallbacks and zero FULL fallbacks, retained in the observed configuration effect.

Table 7. Recorded outcomes and resources on the 87-task ALFWorld reasoning-reviewer panel. Token totals include reused model responses and describe logged work. The same environment-action and recovery budgets apply across all three scopes.

Measure	KEEP	ARG	FULL
Final success	85/87	86/87	87/87
Environment actions	1,336	1,289	1,124
Logged input tokens	221,198	15,925,609	11,880,749
Logged output tokens	42,481	1,924,261	1,261,096
Invalid review replies	0	51	16