SAGA-EDIT: TRACKING NATURAL-LANGUAGE OBLIGATIONS TO REDUCE UNAUTHORIZED DRIFT IN REPOSITORY-LEVEL CODE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

Repository-level code edits can solve an issue while changing behavior the task did not authorize. We study this failure as drift across accepted editing transitions. SAGA-EDIT tracks evidence-backed fix and preserve obligations, checks candidate changes against them, and records decisions in a persistent ledger. On a 104-task subset, the full controller solves 74/104 tasks, with drift in 6/104 trajectories, compared with 68/104 tasks solved and 17/104 with drift without the ledger. Across 168 tasks, full evidence yields 118/168 solved tasks and 11/168 with drift, versus 107/168 and 26/168 with code/test/API evidence alone. In a 12-outcome exploratory family, the corresponding code-only drift contrast has Holm-adjusted $p = .01788$; no-ledger and final-state-only drift contrasts have adjusted $p = .08124$ and $.30859$. A 120-task public-set comparison does not establish solve noninferiority within a 2-percentage-point margin. These results support tracking obligations through a trajectory, while incomplete specifications and limited test coverage preclude a general guarantee of semantic preservation.

1 INTRODUCTION

Repository-level coding agents revise patches after reading issues, inspecting files, and running tests. A passing test suite can establish that a reported defect was repaired for tested inputs, but it cannot establish that every other behavior remained intact. This distinction matters in multi-round editing: a change accepted early in a trajectory may introduce a regression that later visible tests do not exercise. Repository benchmarks evaluate code generation and issue resolution (Hai et al., 2025; Jimenez et al., 2024), while agent systems use tool feedback and repository context to resolve tasks (Yang et al., 2024; Zhang et al., 2024; Xia et al., 2025). Our question is narrower: can an editing controller track what the task authorizes changing and what the available evidence requires it to preserve?

The distinction is familiar from program repair. A plausible patch can satisfy available tests while overfitting them or changing behavior outside the intended repair (Le Goues et al., 2012a; Long & Rinard, 2016; Monperrus, 2018). In repository work, the relevant constraint may be expressed in an issue comment, a review, documentation, a public signature, or a caller rather than in a test. It can also change over time. An explicit migration request may supersede an old interface requirement; an unrelated code edit does not. Final-state validation alone loses the history needed to distinguish those cases.

This creates two coupled errors for an editing agent. It may mistake a description of existing behavior for a request to change it, or it may treat a narrow requested fix as permission for a broader rewrite. Negation and exception clauses are especially consequential because a short phrase can reverse or restrict an obligation. A review thread adds a temporal question: which earlier constraints still apply after the latest instruction? The paper addresses these interpretation problems as part of patch selection, not as a separate natural-language classification benchmark.

A documented parser case makes the problem concrete. The task requests the line number of the second occurrence of a duplicate nested key, while retaining the duplicate-key exception type and message prefix. A candidate that assigns the later value instead of raising an exception can appear to fix line handling but removes the exception behavior. In the case summary, visible checks pass, a preservation probe fails, and the controller rejects that candidate before accepting a localized repair.

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.

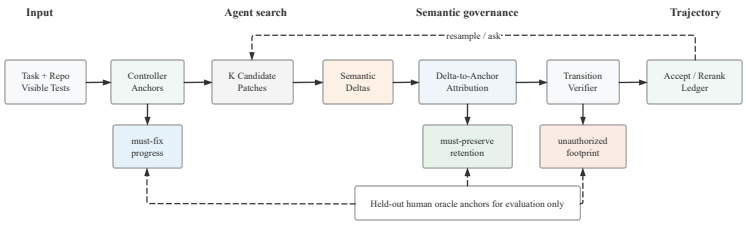


Figure 1: SAGA-EDIT overview. Controller Anchors maps to obligation extraction; Semantic Deltas and Delta-to-Anchor Attribution map to change analysis; Transition Verifier maps to candidate checks; Accept/Rerank Ledger maps to selection and persistent state. Oracle obligations are used for evaluation, not supplied to the controller.

We study SAGA-EDIT, a controller placed between candidate generation and patch acceptance. It records evidence-backed *fix* obligations for requested changes and *preserve* obligations for still-applicable behavior. For each candidate, it extracts behavioral deltas, relates them to the obligations, checks the proposed transition, and retains the decision in a ledger. The ledger permits a later round to revisit an earlier obligation and records explicit supersession instead of silently discarding it. The controller does not claim a complete specification or a proof of semantic preservation.

We evaluate the controller through indexed candidate-sharing comparisons, evidence-source ablations, authorized large edits, and a separate public-task set. The full controller has less observed unauthorized drift than several internal controls. Solve rates generally increase, but the uncertainty and multiplicity analyses do not support every apparent improvement. The evaluation tests an operational way to measure and reduce observed drift within the studied conditions.

2 Problem Setting

At editing round t, an agent has a repository state R_t, task and feedback evidence E_t, and K candidate patches. An *anchor* is a behavior obligation with a type, scope, supporting evidence, and period of applicability. A fix anchor names behavior the task asks to change. A preserve anchor names behavior that remains required unless later evidence explicitly supersedes it. The controller's extracted anchors guide runtime decisions; separately recorded human oracle anchors define evaluation outcomes. These sets need not coincide.

For an active anchor a, let $s_t(a)$ denote its assessed state before a candidate is applied. A fix anchor can be unresolved, partial, or satisfied; a preserve anchor can be violated, at risk, or preserved. Larger $s_t(a)$ indicates greater satisfaction or preservation. Applicability is distinct from state: a completed fix remains relevant to later patches, and an explicit supersession changes which obligation applies. A model's unsupported assertion that an obligation no longer matters is not supersession.

An anchor is necessarily narrower than a full program specification. Its evidence and scope determine which edit it can assess, and an unrecorded obligation cannot contribute to the controller's decision. Conversely, merely mentioning a behavior in source material does not make every related change prohibited. The oracle matching rule likewise requires agreement in obligation type, behavioral meaning, and scope. This distinction helps prevent a high lexical match from being mistaken for a correct interpretation.

For a candidate patch p, let $\Delta(p)$ be the extracted behavior-relevant changes. A delta may alter an API, return value, exception, validation rule, or caller behavior. The intended controller distinguishes progress on active fix anchors from degradation of active preserve anchors and from changes without sufficient authorization. The following raw quantities describe these decision dimensions, with

nonnegative anchor weights w_a and delta risk weights $\omega(\delta)$:

$$\mathrm{APS}(p) = \sum_{a \in A_F} w_a [s_{t+1}(a) - s_t(a)]_+, \tag{1}$$

$$\mathrm{PL}(p) = \sum_{a \in A_P} w_a [s_t(a) - s_{t+1}(a)]_+, \tag{2}$$

$$\mathrm{USF}(p) = \sum_{\delta \in \Delta(p)} \omega(\delta) \mathbf{1}\{\delta \in U\}. \tag{3}$$

Here A_F and A_P are active fix and preserve anchors, $[x]_+ = \max(x, 0)$, and U contains deltas judged unauthorized, unattributed, or insufficiently supported. Authorized progress score (APS), preservation loss (PL), and unauthorized semantic footprint (USF) describe the policy's decision dimensions. These raw quantities are distinct from implementation-level normalization and signal combination (Appendix A). In particular, a low USF is not evidence that every relevant behavior was represented or tested.

We measure *solve* as tasks meeting their task validator, divided by all tasks in a condition. An accepted transition has *oracle drift* if it violates an active oracle preserve obligation without documented supersession. Transition drift divides such transitions by accepted transitions; rejected candidates do not enter its denominator. Task-level *any-drift* counts each task once if any accepted transition drifted. Solve and drift can co-occur: a trajectory can repair the reported defect while introducing an unauthorized regression. On the separate public-task set, the available outcome is a completed regression-test proxy, not the oracle-drift measure used for SB1 and SB3.

3 THE SAGA-EDIT CONTROLLER

The controller first extracts fix and preserve anchors from the current task and repository evidence. It stores each anchor's source, behavioral scope, and active period, including negation and exception clauses in the source wording. Reviews can authorize supersession, but only with external issue or review evidence. Accepted, rejected, and superseded entries remain in the ledger. This design is meant to prevent a later round from treating an earlier obligation as forgotten merely because the latest prompt omits it.

For each candidate, the controller extracts behavior-relevant deltas and attempts to attribute them to active anchors. Attribution asks whether the evidence authorizes a particular behavioral change, not whether the patch is small or textually similar to the issue. The verifier can then use generated tests, static reachability, mutation probes, and entailment evidence to check a proposed transition. A critical preserve conflict without supersession enters the rejection path; otherwise the policy may select, rerank, or withhold a candidate. The ledger records the evidence and decision for the next round. Figure 1 shows the stages.

Conceptual controller round. The controller's intended data flow can be summarized without assuming a particular score normalization or stopping rule:

1. Update active fix and preserve anchors from task, repository, and review evidence; require explicit evidence to supersede an anchor.
2. For each of the K indexed candidates, extract behavior-relevant deltas, attribute them to active anchors, and check the candidate transition.
3. Assess requested progress and preservation risk, flagging critical preserve conflicts without authorization.
4. Accept, rerank, or withhold a candidate under the controller policy; record the evidence, decision, and updated anchor states in the ledger.

The two questions, authorization and observed behavior, should not be conflated. A test may show that a new output occurs, yet leave open whether the task requested it. A review can authorize deleting an old entry point, yet a test is still needed to check the remaining migration behavior. Delta attribution brings the language evidence to the first question; transition checks address the second. Neither signal overrides the other when its underlying evidence is absent.

The parser case illustrates why the stages are distinct. The requested line correction is a fix anchor, whereas the exception behavior is a preserve anchor. Replacing an exception with assignment is a candidate delta. A line-number test can support fix progress, but it does not authorize that delta or test the lost exception path. The reported preservation probe detects the conflict, and the ledger records rejection. The same framework can accept a large migration when a later review explicitly supersedes an obsolete entry point; size alone is not a violation.

Executed controller settings. The authors have verified that the reported settings correspond to the executed protocol. The controller maintains at most 12 active anchors, with cosine-based deduplication at .81 and an entailment threshold of .72. Static reachability is bounded at depth three. Each indexed state has $K = 5$ candidate patches; verification allows at most five generated tests and nine mutation probes per round. The 150-second verifier timeout applies to this check, not to the whole task. The configured PL warning and critical values are .18 and .42; USF acceptance and rerank values are .34 and .21; the APS tie tolerance is .05. These settings bound the search and verification effort. They are not estimates of the probability that a patch preserves all required behavior.

The method depends on the quality of each stage. Missing a preserve obligation, under-extracting a delta, or failing to probe its effect can all admit drift. Conversely, an overly strict interpretation can reject an authorized edit. The comparisons below test the combined controller and selected components under the studied conditions, not the correctness of every intermediate interpretation.

4 EVALUATION

Tasks and controls. We study 352 distinct tasks: SB1 has 168 tasks from 61 repositories, SB3 has 64 authorized-large-edit tasks from 46 repositories, and the public-task set has 120 tasks from 42 repository identifiers. SB3 and SB1 share seven repository identifiers but no task identifiers. The 104-task mechanism comparison spans 52 SB1 repositories and is a subset of the 168-task language comparison. Its full-controller rows reuse the same task and run records; they are not independent replications. The source benchmark identity and version for the public-task set are not documented, so we do not present it as an independently validated benchmark.

Six variants on the 104-task subset separate ledger memory, final-state review, fix and preserve obligations, and hard rejection. Four evidence conditions on all 168 SB1 tasks assess language grounding. The non-ledger controls are internal mechanism comparisons, not official runs of published agents. The candidate-sharing protocol indexes $K = 5$ candidates at each state. Variants begin from the same root state's candidate set; after they accept different patches, they branch into different child states and read those states' indexed candidates. Thus candidate sharing is conditional on repository state, not a claim that divergent states have identical patches.

The 104-task subset includes 69 Python, 19 JavaScript/TypeScript, 10 Java, and six Go tasks. We do not estimate language-specific treatment effects, so this composition is not evidence of cross-language robustness. The larger SB1 condition contains the same 104 full-controller task/run records plus 64 other tasks. We keep subset and full-condition denominators separate throughout the results.

State-indexed replay. The candidate index contains 1,306 state identifiers and 6,530 candidate entries; the branch index records 2,045 accepted transitions across the six mechanism variants. These counts describe the shared state structure, not a single trajectory length. Once variants choose different patches, subsequent states can expose different opportunities to solve the task or incur drift. The appropriate comparison therefore holds the candidate-generation protocol fixed at a given state while allowing selection decisions to change the path. Requiring an identical candidate sequence after divergence would instead ignore the changed repository context.

A missing cache entry terminates the affected branch and is recorded as missing; the protocol does not allow extra generation for just one policy. Shared cache construction and policy replay are distinct sources of cost. Because the results do not include their measured costs or total resource use, we make no cost-efficiency claim. Across all groups, the record contains 1,664 task/condition rows, $104 \times 6 + 168 \times 4 + 64 \times 2 + 120 \times 2$; the overlapping full-controller rows remain one observation per task/run, not additional evidence from repetition.

Table 1: Mechanism variants on the same 104 SB1 tasks. Drift counts use oracle obligations; transition denominators include accepted transitions only.

Variant	Solve	Any-drift	Transition drift
Full controller	74/104	6/104	19/342
No ledger	68/104	17/104	52/339
Final-state-only	70/104	14/104	43/344
Preserve-only	65/104	8/104	25/336
Fix-only	73/104	19/104	61/341
No hard rejection	76/104	15/104	47/343

Labels and analysis. Oracle anchors require matching type, scope, and behavioral meaning; a lexical overlap is insufficient. An ambiguous obligation is kept separate from an explicitly superseded one. Language extraction is scored by micro precision, recall, and F1 against 503 fix and 743 preserve gold anchors per SB1 condition. A matched gold anchor is counted at most once in each condition. Phenomenon labels can overlap and are not additional independent examples. For drift, the unit is an accepted transition or a task, as specified above. SB3's authorized-edit acceptance is a separate task-level outcome; the public-task regression proxy covers all 120 tasks, with every test field reported as completed.

Anchor recovery and task-level outcomes also have different units of analysis. Micro averaging weights individual obligations; tasks with more gold anchors contribute more to the extraction totals. Any-drift, by contrast, weights each task once, regardless of how many anchors or accepted transitions it contains. We therefore do not interpret the anchor count as the number of independent task replications, or use extraction F1 as a substitute for trajectory evaluation.

All six Full-versus-control contrasts are exploratory: three compare mechanism variants on 104 tasks, and one each concerns SB3, the public-task set, and the 168-task language condition. Each has a solve and a drift outcome, forming one family of 12 comparisons. The analysis uses two-sided exact McNemar tests and Holm correction for that family. It also reports task-weighted percentage-point effects with percentile 95% confidence intervals (CIs) from 10,000 repository bootstrap resamples, retaining paired tasks within sampled repositories. These intervals are not multiplicity-adjusted. Table 2 gives the six contrasts and distinguishes raw from corrected values.

Evidence and reproducibility boundary. The authors have checked and approved the execution protocol against the reported experiments. The experimental record summarizes task-level outcomes, anchor matches, risk scores, candidate-state and branch indexes, and three cases. Author verification of the protocol is distinct from an independent replay: aggregate checks alone do not establish identical candidate payloads, evidence, or budgets across variants. Appendix A identifies implementation and provenance fields not reproduced in the written record. The public code and dataset accompany the reported results; neither their availability nor the aggregate comparisons alone establish source-independent benchmark validity.

5 Results

5.1 Persistent Obligations and Candidate Decisions

Table 1 shows the 104-task comparison. The full controller solves 74/104 tasks and has any-drift on 6/104, with 19 drift-positive accepted transitions among 342. Removing the ledger changes these outcomes to 68/104 solve and 17/104 any-drift. Final-state-only review also has higher any-drift, 14/104, than the full controller. This pattern is consistent with a benefit from retaining applicable obligations across rounds, although the indexed protocol alone cannot certify equal candidate payloads after branching.

The remaining variants expose a tradeoff rather than a single ranking. Relative to preserve-only, the full controller solves nine more tasks, while the any-drift difference is only two tasks. Fix-only nearly matches full solve but has 19 drift tasks. Disabling hard rejection adds two solved tasks and nine drift

Table 2: Exploratory paired effects in percentage points (pp). Positive effects favor Full: solve gains or drift reductions. Raw and Holm p values appear only where reported in the aggregate results. Public-task drift is the regression proxy.

Comparison	Outcome	Full effect [95% CI]	Raw p	Holm p
No ledger (104)	Solve	$+5.77$ $[-0.95, 12.65]$	.14600	–
	Any-drift	$+10.58$ $[3.74, 17.39]$	.00739	.08124
Final-state-only (104)	Solve	$+3.85$ $[-1.98, 9.74]$	.34375	–
	Any-drift	$+7.69$ $[1.90, 13.73]$	.03857	.30859
Preserve-only (104)	Solve	$+8.65$ $[2.61, 14.85]$	.02246	–
	Any-drift	$+1.92$ $[-1.80, 5.83]$	.62500	–
SB3 non-ledger (64)	Solve	$+7.81$ $[-1.41, 17.46]$	.17969	–
	Any-drift	$+12.50$ $[3.12, 22.81]$	.02148	–
Public non-ledger (120)	Solve	$+1.67$ $[-2.16, 6.00]$	.72656	–
	Regression proxy	$+6.67$ $[0.98, 12.10]$	.05737	–
Code/test/API-only (168)	Solve	$+6.55$ $[0.00, 13.33]$	.05224	–
	Any-drift	$+8.93$ $[3.73, 14.57]$	–	.01788

tasks. These outcomes motivate tracking both requested progress and preserved behavior; they do not isolate the effect of every verifier component.

Accepted-transition counts give a complementary view. Full has 19/342 drift-positive accepted transitions, compared with 52/339 without the ledger and 43/344 under final-state-only review. These denominators differ because the variants accept different paths, so a transition-level percentage is not a paired per-task effect. Any-drift instead gives each task one outcome and is the basis of the paired contrasts. The two measures agree in direction for these variants, but they answer different questions about trajectory quality.

The estimated any-drift reductions are 10.58 percentage points versus no ledger and 7.69 points versus final-state-only. Their repository bootstrap intervals exclude zero, but the corresponding exact drift tests have Holm-adjusted $p = .08124$ and .30859 across the 12 exploratory outcomes. We therefore treat these as descriptive mechanism evidence, not multiplicity-corrected discoveries. Solve gains versus those controls have intervals that include zero (Table 2).

Reading the paired comparison. For no ledger, 65 tasks are solved by both systems, three by the control alone, nine by Full alone, and 27 by neither. The solve gain is therefore $100(9-3)/104 = 5.77$ pp. The corresponding drift comparison has four tasks drifting under both systems, 13 under the control alone, two under Full alone, and 85 under neither, yielding a reduction of $100(13-2)/104 = 10.58$ pp. The exact test uses these discordant pairs rather than just the marginal rates. Repository bootstrapping instead resamples whole repositories while retaining paired tasks. Its interval and the corrected exact test answer related but different inferential questions; a favorable interval does not supersede the family-adjusted result.

5.2 Language Evidence and Obligation Recovery

On all 168 SB1 tasks, full evidence yields 1,111 true positives (TP), 130 false positives (FP), and 135 false negatives (FN) in anchor matching, for a micro F1 of .893. The code/test/API-only condition has F1 of .786 and lower recall for both fix and preserve anchors. It also solves 107 rather than 118 tasks and has any-drift on 26 rather than 11 tasks. The full controller's 104-task row above is a subset of these same full-evidence run records, not an additional experiment.

Full-evidence precision and recall are .895 and .892; the code/test/API-only values are .836 and .742. The gap is therefore not explained solely by producing more anchors. Removing issue/review evidence yields F1 of .802, whereas removing docs/comments yields .853, with different fix and preserve losses in each condition. The type-specific counts in Table 4 show which obligations are missed; the aggregate F1 alone would hide that distinction.

The full-versus-code/test/API-only any-drift difference is 8.93 percentage points, with a repository interval of [3.73, 14.57] points and Holm-adjusted $p = .01788$. Its solve gain is 6.55 points, with an interval of [0.00, 13.33] and raw exact $p = .05224$; the lower interval endpoint and unadjusted test do

Table 3: SB1 evidence conditions on 168 tasks and 1,246 oracle anchors per condition. F1 is micro-averaged; any-drift is task-level.

Evidence	TP/FP/FN	F1	Solve	Any-drift
Full	1111/130/135	.893	118/168	11/168
Code/test/API-only	925/182/321	.786	107/168	26/168
No issue/review	936/152/310	.802	109/168	23/168
No docs/comments	1028/135/218	.853	115/168	18/168

Table 4: Fix/preserve detail on 168 SB1 tasks. Recall entries are matched/gold; FP counts extra extracted anchors. Transition drift uses accepted transitions only.

Evidence	Fix recall	Preserve recall	Fix/Preserve FP	Transition drift
Full	459/503	652/743	47/83	31/550
Code/test/API-only	343/503	582/743	71/111	70/547
No issue/review	354/503	582/743	64/88	66/549
No docs/comments	455/503	573/743	43/92	48/553

not justify a general solve claim. Removing issue/review or documentation/comment evidence also reduces anchor F1 and raises observed drift. These are condition-level associations because changing the evidence can alter both extracted anchors and later decisions.

Every evidence condition uses the same 503 fix and 743 preserve gold anchors. Full evidence recovers 459/503 fix anchors and 652/743 preserve anchors (Table 4). Relative to code/test/API-only evidence, fix recall is 23.06 percentage points higher and preserve recall is 9.42 points higher. These differences describe which obligations are recovered; they are not independent experiments from the downstream solve and drift comparisons.

The linguistic slices clarify what is at stake. Among gold anchors tagged for negation, exception constraints, and review supersession, full evidence recovers 57/63, 40/47, and 28/35, compared with 46/63, 31/47, and 18/35 for code/test/API-only evidence. Labels can overlap, so their denominators must not be summed. Negation determines whether behavior is required or prohibited; exception wording restricts the inputs on which a rule applies; review supersession changes which earlier obligation remains active. Recovering these distinctions matters for authorization even when the resulting code edit is small.

Each phenomenon poses a different interpretation problem. Dropping a negation can reverse the direction of an obligation even when the remaining words match the source. Dropping an exception can extend a requirement beyond its intended inputs. Ignoring review order can preserve a superseded requirement or prematurely discard one that still applies. The controller therefore retains polarity, scope, and evidence order with the obligation rather than treating an anchor as a keyword label. The reported slice counts measure correct obligation recovery under these distinctions; they do not measure each possible error separately or establish which extraction component caused the improvement.

The source ablations also caution against treating language as redundant commentary on code. Removing issue/review evidence predominantly reduces fix recovery in these counts, whereas removing docs/comments leaves fix recovery close to Full but reduces preserve recovery. This pattern is consistent with the different roles of change requests and descriptions of existing behavior. It does not imply that either source always has that role: an explicit later review may authorize a removal, while documentation can itself describe a requested migration.

5.3 Authorized Edits and Public-Task Checks

SB3 tests whether preservation checks also permit explicitly authorized large edits. The full controller accepts 58/64 authorized edits, compared with 53/64 for its matched non-ledger control. It solves 48/64 rather than 43/64 tasks and has any-drift on 4/64 rather than 12/64. The solve-gain interval includes zero, whereas the drift-reduction interval is [3.12, 22.81] percentage points. These are

Table 5: Matched conditions on SB3 and the public-task set. The public outcome is a completed regression-test proxy, not SB3 oracle drift. Authorized acceptance applies only to SB3.

Set and system	Solve	Drift/proxy	Authorized accept
SB3, full	48/64	4/64	58/64
SB3, non-ledger	43/64	12/64	53/64
Public-task set, full	64/120	11/120	n/a
Public-task set, non-ledger	62/120	19/120	n/a

exploratory comparisons, not evidence that the controller will generalize to every migration or repository.

Authorized acceptance and solve are deliberately separate outcomes. Accepting a permitted migration does not establish that every part of the task was completed, just as completing the requested fix does not establish an absence of drift. We report their marginal counts without inferring an unreported intersection. Together they test whether preservation checks merely block broad changes or can distinguish permission to change one behavior from obligations that remain in force.

The public-task set gives a smaller solve difference, 64/120 versus 62/120, and fewer regression-proxy failures, 11/120 versus 19/120. Its solve-gain interval is [-2.16, 6.00] percentage points, so the available data do not establish noninferiority within a 2-percentage-point margin. The exact raw p for the regression proxy is $.05737$; the bootstrap interval and exact test provide different uncertainty summaries, both of which we report. Without a documented source benchmark and exposure history, the public-task result is a check on these 120 tasks rather than evidence of generalization to an independently defined benchmark.

5.4 USF as a Risk Signal

The USF evaluation covers 1,487 patches, including 206 labeled unauthorized. At the .34 working point, USF detects 151 unauthorized patches and misses 55. At comparison working points chosen to match this 151/206 recall, static-risk and patch-size scores yield more false positives (Table 6). This comparison concerns one classification operating point, not area under a curve, probability calibration, or a deployment threshold validated on an independent development set.

Table 6: Unauthorized-patch classification at matched recall (.733). TN denotes true negatives. Comparator thresholds are descriptive working points on the test set.

Signal	TP	FP	FN	TN	Precision	Recall
USF $\geq .34$	151	37	55	1244	.803	.733
Static-risk $\geq .57$	151	92	55	1189	.621	.733
Patch-size ≥ 96	151	126	55	1155	.545	.733

USF has 89 fewer false positives than patch size at this matched recall, but the common recall is a selection condition rather than an independent result. Moreover, the accepted token-refresh case in Section 5.5 shortened an untested grace-period fallback: it solved the reported task while retaining unauthorized drift. The case illustrates the residual dependence on anchor coverage and probing, not a measured frequency of this failure mode.

The record does not specify when the USF cutoff was selected or whether the runtime acceptance guard used the same scale. The classification table therefore supports a comparison at these operating points; the trajectory tables remain the direct evidence about accepted edits.

5.5 Decisions and Failure Cases

The three recorded cases connect task evidence, controller decisions, and oracle outcomes. They include a rejected regression, an accepted regression, and an authorized removal. These are illustrative summaries rather than a random sample of failures, so they cannot establish the prevalence of a mechanism or replace the aggregate comparisons.

Duplicate nested-key diagnosis (CASE-01, SB1-007). The task requests a corrected duplicate-key line number while retaining the exception type and message prefix. A candidate substitutes assignment for the exception. Visible checks pass, but a preservation probe fails. The controller rejects that candidate and accepts a localized correction. The task is solved after four accepted transitions, with no oracle drift. Here the deciding evidence is not the size of the patch: the assignment removes behavior that the task still requires. The preservation check supplies evidence that a fix-oriented test misses.

Token-refresh grace period (CASE-02, SB1-013). A token-refresh migration changes an expiry check from `expires_at+grace` to `expires_at`. Ordinary checks pass, but the grace-period boundary is not covered. The patch is accepted with the unauthorized regression still present. The task is solved after five accepted transitions, two of which are drift-positive. This outcome separates solving the stated task from preserving other required behavior. A persistent ledger cannot compensate for a boundary condition that is not adequately represented or checked.

Authorized entry-point removal (CASE-03, SB3-002). A later review explicitly authorizes removing an obsolete entry point after migration, while preserving environment-variable precedence. Caller and preservation checks pass, and the controller records supersession. The authorized edit is accepted and the task is solved after six accepted transitions, with no oracle drift. In contrast to the first case, removing old behavior is permitted by later evidence. The remaining preservation obligation still applies, so supersession is local to the obsolete entry point rather than permission to discard unrelated constraints.

6 Related Work

Tool-using language agents interleave reasoning and action (Yao et al., 2023). Repository agents combine code generation, localization, and test feedback (Yang et al., 2024; Zhang et al., 2024; Xia et al., 2025; Ruan et al., 2025). Plan/edit/verify and code-test coevolution systems likewise make intermediate evidence important (Tarshish et al., 2026; Li et al., 2026). Trajectory analyses examine behavior obscured by final outcome metrics (Mehtiyev & Assunção, 2026). Our internal non-ledger and static-risk controls compare mechanisms within this study; they are not official implementations or leaderboard comparisons against those papers.

Program repair and generated testing motivate the distinction between a plausible test-passing patch and a behaviorally appropriate one (Le Goues et al., 2012b; Nguyen et al., 2013; Just et al., 2014; Fraser & Arcuri, 2011; Jia & Harman, 2011). Contracts, behavioral subtyping, and refactoring work motivate attention to behavior that must remain stable (Hoare, 1969; Meyer, 1992; Liskov & Wing, 1994; Fowler et al., 1999). SAGA-EDIT does not infer a complete contract. Its contribution is to make partial, language-grounded obligations and their temporal status explicit in a candidate-selection workflow, then evaluate accepted-transition drift separately from final solve.

7 Conclusion

The comparisons associate persistent fix/preserve obligations with less observed unauthorized drift, while retaining or improving solve in several conditions. The 104-task mechanism subset, 168-task language comparison, SB3 authorized edits, and 120 public tasks answer different questions and must not be pooled as independent confirmations. Under the exploratory 12-outcome analysis, some apparent gains remain uncertain; the public solve result does not establish noninferiority. The practical lesson is to audit what each accepted edit changes against still-applicable evidence, while treating incomplete obligations and tests as limits on any preservation claim.

The controller can also be overly conservative: existing behavior may be a bug rather than a preservation requirement. Explicit supersession helps only when the task evidence is clear. Weak documentation and missing boundary cases remain failure modes, and the current results do not quantify their prevalence.

AI USE STATEMENT

Generative AI assisted experimental planning, methodological discussion, interpretation of reported results, literature and citation checking, and manuscript preparation, including language editing and LaTeX formatting. The reported experimental outcomes are researcher-provided observations, not AI-generated predictions. The use of language models within the editing controller is described separately in the method. The authors retain responsibility for the final text, claims, references, and artifacts, including all AI-assisted content.

ETHICS STATEMENT

This study concerns code editing in repositories and the preservation of task-authorized behavior. Its intended use is to help developers inspect changes, not to replace developer review or provide a security guarantee. Incorrectly extracted obligations can preserve bugs, miss intended behavior, or reject necessary changes; passing the reported checks does not establish that a patch is safe to deploy. The authors confirm compliance with the ICLR Code of Ethics and applicable requirements for privacy, licensing, attribution, and consent, and confirm that all required approvals and permissions for the study and released materials have been obtained.

REPRODUCIBILITY STATEMENT

The implementation and dataset are publicly available in the code repository[1] and dataset repository[2], respectively. The authors have verified and approved the reported execution protocol and its correspondence with the experiments. The method, evaluation protocol, and appendix specify controller settings, task groups, metric denominators, and paired statistical analysis. Appendix A distinguishes those confirmed settings from implementation and provenance details not reproduced in the written record. Author verification and public availability are not presented as independent reproduction of the results.

REFERENCES

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

Gordon Fraser and Andrea Arcuri. EvoSuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 416–419, 2011. doi: 10.1145/2025113.2025179.

Nam Le Hai, Dung Manh Nguyen, and Nghi D. Q. Bui. On the impacts of contexts on repository-level code generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1496–1524, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.82. URL https://aclanthology.org/2025.findings-naacl.82/.

C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580, 1969. doi: 10.1145/363235.363259.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. doi: 10.1109/TSE.2010.62.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 437–440, 2014. doi: 10.1145/2610384.2628055.

[1] https://anonymous.4open.science/r/saga-edit-artifact-2027/

[2] https://huggingface.co/datasets/blindpeerbench/sagaedit

Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Proceedings of the 34th International Conference on Software Engineering*, pp. 3–13, 2012a. doi: 10.1109/ICSE.2012.6227211.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012b. doi: 10.1109/TSE.2011.104.

Kefan Li, Yuan Yuan, Mengfei Wang, Shihao Zheng, Wei Wang, Ping Yang, Mu Li, and Weifeng Lv. Beyond fixed tests: Repository-level issue resolution as coevolution of code and behavioral constraints. arXiv preprint arXiv:2604.04580, 2026. URL https://arxiv.org/abs/2604.04580.

Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994. doi: 10.1145/197320.197383.

Fan Long and Martin Rinard. An analysis of the search spaces for generate and validate patch generation systems. In *Proceedings of the 38th International Conference on Software Engineering*, pp. 702–713, 2016. doi: 10.1145/2884781.2884872.

Tural Mehtiyev and Wesley Assunção. Beyond resolution rates: Behavioral drivers of coding agent success and failure. arXiv preprint arXiv:2604.02547, 2026. URL https://arxiv.org/abs/2604.02547.

Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992. doi: 10.1109/2.161279.

Martin Monperrus. Automatic software repair: A bibliography. *ACM Computing Surveys*, 51(1):17:1–17:24, 2018. doi: 10.1145/3105906.

Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. SemFix: Program repair via semantic analysis. In *Proceedings of the 35th International Conference on Software Engineering*, pp. 772–781, 2013. doi: 10.1109/ICSE.2013.6606623.

Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. SpecRover: Code intent extraction via LLMs. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ICSE '25, pp. 963–974. IEEE Press, 2025. ISBN 9798331505691. doi: 10.1109/ICSE55347.2025.00080. URL https://doi.org/10.1109/ICSE55347.2025.00080.

Noam Tarshish, Nofar Selouk, Daniel Hodisan, Bar Ezra Gafniel, Yuval Elovici, Asaf Shabtai, and Eliya Nachmani. SAFEdit: Does multi-agent decomposition resolve the reliability challenges of instructed code editing? arXiv preprint arXiv:2604.25737, 2026. URL https://arxiv.org/abs/2604.25737.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 1592–1604, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

A Protocol and Reporting Detail

The authors confirm that the controller settings and execution protocol in the main text were applied to the reported experiments. The design identifies a `core-v2` materials configuration; that identifier is not an original run commit or release version. Author confirmation resolves the status of the reported protocol, but does not supply implementation fields absent from the written description or constitute an independent replay.

The described candidate index has neither patch payloads nor original evidence hashes. Its state and branch counts permit structural checks but not a content-level cache audit from the written summary alone. Each condition contains one run per task, with no aggregation across repeated runs. The model seed is recorded as not applicable; the processing seed is 9262026. These are separate from the repository resampling procedure used for uncertainty estimates.

The annotation rule treats a fix or preserve anchor as a behavior with scope and evidence. A later issue or review can explicitly supersede an anchor; ambiguity alone does not. Matching requires type, scope, and behavioral meaning. Unmatched additional controller anchors count as false positives, and a gold anchor can be matched only once per condition. For task-level any-drift, one or more violating accepted transitions produce one positive task. SB3 authorized acceptance has a 64-task denominator. Public-task regression-proxy fields are completed on all 120 tasks; they are not coded as zero when a test was absent.

The written record does not reproduce the exact model and role assignments, decoding settings, prompt text, input truncation, numeric state encoding, score normalization, signal combination, overall call and time limits, or stopping rule. The annotator count and blind-labeling process are not specified there, nor are the public-task source benchmark, version, freeze time, and tuning exposure. These are documentation limits, not a statement that the confirmed protocol was unexecuted. Full procedural reproduction requires those fields in addition to the reported settings and author confirmation.

The cases in Section 5.5 link controller anchors, verifier summaries, decisions, and oracle labels. The written case records do not include original source commits, raw evidence locations, or machine verifier outputs. We therefore use them to explain decisions and outcomes, not as independently replayed traces.

B Paired Contrasts and Uncertainty

Table 2 in the main text reports all six Full-versus-control contrasts. A positive solve effect is a gain for Full; a positive drift effect is a reduction for Full. Intervals are task-weighted, repository-bootstrap percentile 95% intervals from 10,000 resamples within each contrast. The tests are exact, two-sided paired McNemar tests. The family consists of all 12 solve and drift comparisons. Dashes in the table indicate values not reported in the aggregate results, not values of zero or nonsignificance. The intervals are not multiplicity-adjusted.

The public solve interval includes effects worse than the proposed 2-percentage-point noninferiority margin. No noninferiority conclusion follows from a positive point estimate. Similarly, the public regression-proxy raw $p = .05737$ remains the stated exact-test result even though its bootstrap interval excludes zero. Reporting both avoids selecting only the favorable summary.