WHY PREDICT WHAT YOU CAN PRESERVE? CONTROLLING GENERATIVE SCOPE IN LLM STATE UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly modify persistent code and structured state, yet even a small requested change is often delivered as an entire rewritten successor. This makes the model regenerate source content that is already known, exposing untouched fields, delimiters, and code to omissions or incidental changes. Output validation can ensure a well-formed candidate without ensuring fidelity of unchanged state; compact edit formats restrict model-authored content but can fail at address resolution or value encoding. We therefore treat preservation as a relation between an authoritative source and its successor, rather than as a property of an output in isolation. We define *generative scope* as the source region whose successor content the model must provide, and *generative exposure* as the unchanged portion of that region. Our method binds a supplied edit to the source, allocates model versus runtime responsibility for the successor, and composes accepted local replacements with unchanged source content. Under sound source and address binding, this composition preserves the untouched frame by construction, while leaving replacement correctness and applicability as separate obligations. This formulation makes preservation and edit-format failures measurable without asserting a model success law. With the correct replacement and address supplied, local Patch improves byte-exact fidelity over full Rewrite by 40.0 and 39.1 percentage points on QuixBugs (40 tasks) and all eligible HumanEvalFix-Python edits (133 tasks); released-test gains are only 2.5 and 0.8 points.

1 INTRODUCTION

An LLM asked to change one configuration value may return an entire document. An agent repairing one expression may regenerate its surrounding file. In both cases, most of the requested output is already present in the source. The interface nevertheless makes the model responsible for reproducing that content: a dropped field, changed delimiter, or altered line ending can become part of the successor. As agents increasingly operate on persistent software and structured environments (Yang et al., 2024; Wang et al., 2025; Yao et al., 2025), this raises a basic design question: *which parts of a state update should be generated, and which should be preserved directly?*

The distinction concerns the model's output obligations. Autoregressive generation conditions each emitted token on its prefix; seeing source content in the input does not enforce equality in the output. A runtime can instead retain that content exactly while installing an update. We call the returned source region *generative scope*; its unchanged portion is *generative exposure*, existing state whose preservation now depends on model output. Exposure excludes address and response-format overhead, so smaller scope need not mean fewer output tokens or monotonically higher reliability.

Language-model systems delegate precise operations to external computation (Gao et al., 2023; Zhang et al., 2025), while editing systems reuse source spans or apply generated changes (Panthaplackel et al., 2021; Novikov et al., 2025). We examine this division of work at the *state-update boundary*: paired arms receive the same source and edit, but differ in who writes unchanged successor content. This isolates an output obligation that prior editing systems often optimize jointly with repair discovery.

This question requires separating output validity from state fidelity. Schema-conforming output may contain incorrect values (OpenAI, 2024); even a well-formed, executable candidate can alter content

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.

that an edit should preserve. Preservation is a *relation* between the authoritative source and candidate, rather than a property of the candidate in isolation. Semantic generation constraints and verified update boundaries are established research directions (Nagy et al., 2026; Bai & Cai, 2026). We study the specific relation that the untouched source frame must survive an update. Under sound source and address binding, deterministic composition enforces this relation for every applicable local payload, independently of whether the payload itself is correct.

To measure this tradeoff, we supply each interface the correct replacement text and its source address. A deterministic executor can therefore install the edit without a model; this is the natural oracle upper bound. Generated encodings can nonetheless have different delivery success, as noted for diffs and full-file outputs (Anthropic, 2024). Our evaluation setup isolates expressing and applying an already known edit, counts rejected outputs as failures, and distinguishes applicability, byte fidelity, and released-test outcomes. It does not measure autonomous repair discovery.

Overall, our contributions are summarized as follows:

- We identify a source-relative failure mode in fixed-intent delivery: an applicable, test-passing generated file can differ in untouched content. We measure this against direct deterministic application of the supplied edit.
- We formulate *generative scope* and *generative exposure* and organize edit delivery into source binding, scope control, and composition. An aligned local splice satisfies a structural frame-preservation invariant, while address, value, and encoding failures remain possible.
- We evaluate paired delivery paths on binary, JSON, QuixBugs, and HumanEvalFix-Python tasks. Local Patch attains 40/40 and 133/133 byte-exact files versus 24/40 and 81/133 for full Rewrite; released-test differences are only one task per set, while supplementary studies expose encoding and applicability limits.

2 Related Work

2.1 Source reuse and the division of generative work

Edit-based generation reuses source content instead of generating every successor token. LaserTagger and FELIX combine token tagging with insertion; Seq2Edits predicts keep or replace operations over spans (Malmi et al., 2019; Mallinson et al., 2020; Stahlberg & Kumar, 2020). Panthaplackel et al. copy source spans in one step (Panthaplackel et al., 2021), while AlphaRepair localizes code generation by masked completion (Xia & Zhang, 2022). AlphaEvolve searches over model-generated code changes, while Cascaded Code Editing separates edit sketches from their application (Novikov et al., 2025; Wang et al., 2026).

PAL delegates execution to an interpreter, while Recursive Language Models expose long inputs through a programmable environment (Gao et al., 2023; Zhang et al., 2025). We study the output side of this division: which unchanged source regions must be regenerated under a supplied edit? We operationalize generative scope at fixed intent. Our paired paths receive the same full source; they do not test long-context retrieval or autonomous repair discovery.

2.2 Output constraints and source-relative preservation

Structured Outputs constrains generation to a supplied schema while leaving errors within generated values possible (OpenAI, 2024). Controlled generation also extends beyond syntax: sequential Monte Carlo incorporates syntactic and semantic constraints, and ChopChop constrains programs for type safety or equivalence to a reference (Loula et al., 2025; Nagy et al., 2026). Closely related runtime designs address source-relative effects. PatchOptic combines projected reads, authorized writes, and verified updates; Cordon stages tool effects in semantic transactions; Shepherd exposes reversible execution traces (Bai & Cai, 2026; Chen et al., 2026; Yu et al., 2026).

PatchOptic addresses authorization and verifiable tool effects, whereas our conditional frame law concerns what a generated successor must reproduce. The law follows from a sound splice; its contribution here is an explicit endpoint and a paired measurement of preservation against format failures, not a new semantic-equivalence guarantee.

2.3 EDITING INTERFACES AND FIXED-INTENT EVALUATION

Equivalent edits can impose different format demands on a model, as Anthropic's agent-interface study observes for diffs and full-file rewrites (Anthropic, 2024). SWE-agent studies agent–computer interfaces; CODESTRUCT exposes syntax-tree actions (Yang et al., 2024; Kim et al., 2026). Diff-XYZ evaluates diff understanding, AdaEdit chooses between structural diffs and full code, and SWE-Edit separates code viewing and editing into specialized subagents (Glukhov et al., 2025; Cheng et al., 2026; Zhang et al., 2026). Agentless combines localization, repair, and validation (Xia et al., 2025).

End-to-end benchmarks such as SWE-bench and τ-bench also require action discovery (Jimenez et al., 2024; Yao et al., 2025). Our correct replacement and address remove that discovery step: direct application is an oracle control, while model calls measure delivery loss. We therefore adapt only the named systems' single edit actions or grammars; their published repair scores do not estimate this conditional effect.

3 PRELIMINARIES: GENERATION AND STATE PRESERVATION

A source-relative update. Let $S \in \mathcal{S}$ be authoritative state, I^* a supplied correct intent, $W \subseteq \Omega(S)$ its writable footprint, and $S^* = \Phi(S, I^*)$ its canonical successor. A descriptor D names source identity, addresses, and old values. Mechanical applicability $\mathrm{Chk}(D, S)$ checks version, values, and nonoverlap; semantic $\mathrm{Align}(D, S, I^*)$ additionally requires the *intended* boundary. Frozen tasks use the aligned D^*; insertions need an explicit anchor even if their old value is empty. With replacement-transported boundaries $D^*[z]$, code targets obey

$$\pi_{D^*[z^*]}(S^*) = z^*, \qquad F_{D^*[z^*]}(S^*) = F_{D^*}(S), \tag{1}$$

where F is the ordered untouched frame (a complement projection for bits and fixed-topology JSON). Code behavior is scored separately.

A testable preservation relation. Let $\mathrm{put}_D(S, z)$ replace exactly the spans named by D and copy their complement from S; $\mathcal{Z}_D$ contains syntactically representable replacement values. Define the frame-admissible family

$$\mathcal{C}_D(S) = \begin{cases} \{\mathrm{put}_D(S, z) : z \in \mathcal{Z}_D\}, & \mathrm{Chk}(D, S), \\ \varnothing, & \text{otherwise.} \end{cases} \qquad U_D(S') = \mathbf{1}[S' \in \mathcal{C}_D(S)]. \tag{2}$$

Membership tests preservation, not value correctness; a full rewrite may satisfy it without enforcing it. A task-fixed witness resolves repeated code, and we score U_{D^*} rather than a post-hoc descriptor.

Generative exposure. For interface d, let B_d be the source region whose successor content the model must supply. Under the intended binding, $W \subseteq B_d \subseteq \Omega(S)$, and

$$\Gamma_d = B_d \setminus W, \qquad \Omega(S) = W \,\dot\cup\, \Gamma_d \,\dot\cup\, (\Omega(S) \setminus B_d). \tag{3}$$

These are edited, model-regenerated unchanged, and runtime-carried regions. Insertions retain a boundary anchor. Exposure excludes address syntax and is measured in source bits, fields, or characters, not output tokens.

Proposition 1 (Scope containment and local frame preservation)**.** *If a mechanically applicable resolver replaces only B_d, inserts only at source boundaries covered by that region's binding, and copies $\Omega(S) \setminus B_d$ from authoritative S, it preserves all source content outside B_d. When $\mathrm{Align}(D, S, I^*)$ holds, any collateral change relative to W is confined to Γ_d. If the local $B_d = W$ resolver also follows D's boundaries and uses put_D, every applicable candidate belongs to $\mathcal{C}_{D^*}(S)$. These statements impose no ordering on model success probabilities.*

Proof. The resolver copies every segment outside B_d by construction. Since $W \subseteq B_d$, source content outside W can change only inside $B_d \setminus W$. Under the stated local splice and alignment premises, the candidate lies in the defining image of put_{D^*}, including when an insertion consumes no source atom. Appendix C handles transported boundaries and multi-span updates. □

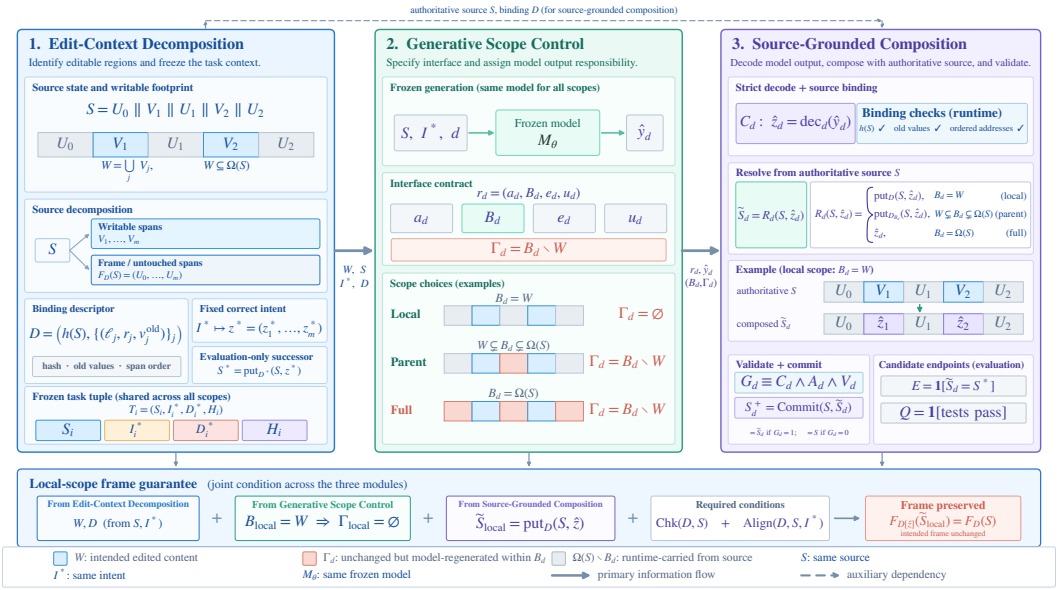


Figure 1: Three modules control generative scope: **Edit-Context Decomposition** fixes the source-bound intent, **Generative Scope Control** assigns model-authored region B_d and unchanged exposure Γ_d, and **Source-Grounded Composition** decodes, validates, and composes the update. Under aligned intent and valid binding, a local splice preserves the untouched frame. Code E and Q score candidates.

4 Method: Controlling Generative Scope

We instantiate the framework with *Edit-Context Decomposition*, *Generative Scope Control*, and *Source-Grounded Composition* (Figure 1). They respectively fix an edit boundary, assign output responsibility, and construct and assess the successor. The scope is specified by an interface; this method does not learn a scope selector.

4.1 Edit-Context Decomposition

Freeze $\mathcal{T}_i = (S_i, I_i^*, D_i^*, H_i)$ before inference. The supplied correct intent includes replacement value z_i^*; its descriptor D_i^* records source identity $h(S_i)$, ordered nonoverlapping writable spans, and expected old values; H_i specifies structural invariants. The submitted D_i equals D_i^* in the main fixed-intent tasks. For code, $S = U_0 \| V_1 \| U_1 \| \cdots \| V_m \| U_m$ separates writable V_j from untouched U_j. A source-bound splice is

$$\mathrm{put}_D(S, z) = U_0 \| z_1 \| U_1 \| \cdots \| z_m \| U_m, \qquad S^* = \mathrm{put}_{D^*}(S, z^*). \tag{4}$$

Bits use indexed replacement; JSON uses addressed subtree replacement. The complete S^* is withheld from the model, but D^* and z^* suffice for deterministic application without inference. The runtime checks $\mathrm{Chk}(D, S)$ before applying an action, while the guarantee relative to the intended footprint additionally needs $\mathrm{Align}(D, S, I^*)$; hash and old-value checks alone do not prove alignment. A zero-length insertion needs a boundary witness beyond its empty old value (Proposition 5); changed code lengths use transported boundaries (Appendix C).

4.2 Generative Scope Control

An interface $r_d = (a_d, B_d, e_d, u_d)$ prescribes address resolution, returned successor scope, encoding, and ownership of unchanged-state carry-forward. The same frozen M_θ receives the full source and correct intent under its interface prompt $\mathcal{P}_d(S, I^*)$ and emits $\hat{y}_d$; a strict dec_d produces payload $\hat{z}_d$ or rejects it. Let $V_d(y)$ be the declared gate outcome for the candidate decoded from y. Each interface admits a set of successful responses

$$\mathcal{L}_d(S, I^*) = \{y : \mathrm{dec}_d(y) \neq \bot,\ A_d(y) = 1,\ R_d(S, \mathrm{dec}_d(y)) = S^*,\ V_d(y) = 1\}. \tag{5}$$

Where an eligible gold action exists, it belongs to this set; equal targets do not imply equal model success across interfaces. For accepted, applicable responses, the controlled scope resolvers construct

$$\widetilde{S}_d = R_d(S, \hat{z}_d) = \begin{cases} \mathrm{put}_D(S, \hat{z}_d), & B_d = W \quad \text{(local)}, \\ \mathrm{put}_{D_{B_d}}(S, \hat{z}_d), & W \subset B_d \subset \Omega(S) \quad \text{(parent)}, \\ \hat{z}_d, & B_d = \Omega(S) \quad \text{(full state)}. \end{cases} \tag{6}$$

Here D_{B_d} binds the larger returned region and preserves its complement; rejected contracts or failed source checks yield $\perp$. A compact local encoding can keep checked D in the runtime and ask the model only for z, changing e_d without changing B_d. All arms may read the full source, while Γ_d specifies which unchanged successor content the model must write. Published tool and edit-format paths can also change a_d or e_d, so their contrasts estimate complete interface effects. Matched substudies isolate selected responsibilities; a local contract may even use more output tokens than a full rewrite.

4.3 Source-Grounded Composition

Let C_d mean accepted response format, A_d an applicable candidate after source checks, and V_d passage of the declared gate. On stateful tasks the final transition is

$$S_d^+ = \mathrm{Commit}(S, \widetilde{S}_d) = \begin{cases} \widetilde{S}_d, & C_d \wedge A_d \wedge V_d, \\ S, & \text{otherwise}. \end{cases} \tag{7}$$

Local composition with $\mathrm{Align}(D, S, I^*)$ makes $U_d = 1$ for every applicable candidate, yet a wrong payload may leave $T_d = 0$. A parent splice preserves source outside its parent; a full rewrite has no corresponding copied region. We distinguish C, A, T, U, V, exact target E, and released-test pass Q, reporting only the quantities observable in each study. Program results score isolated candidates (E, Q), with any method-specific rejection included in A and no common persistent commit. Appendix C derives the exact-success tradeoff and specifies when the structural preservation guarantee improves total reliability.

5 Experiments

We evaluate three questions: how delivery paths with the same intended update differ in state fidelity; when enlarging generative exposure changes success; and where source-grounded preservation ceases to apply. Main comparisons measure complete interfaces, while matched substudies isolate selected responsibilities.

5.1 Datasets, interfaces, and endpoints

We test one-bit transitions, type-preserving edits in 40 public JSON files, opaque-scalar and lockfile-list payloads, and fixed Python edit delivery on 40 QuixBugs plus 40 hash-selected HumanEvalFix-Python tasks, then all 133 eligible HumanEvalFix tasks (Lin et al., 2017; Muennighoff et al., 2024). Appendix Table 3 specifies task construction.

Baselines are *delivery interfaces*: Patch/leaf returns a local operation; parent/full Rewrite materializes larger successors. Paired arms share source, correct replacement, and address; invalid, timed-out, and missing outputs remain failures. We distinguish contract C, applicability A, intended edit T, frame preservation U, gate V, canonical target E, and released-test pass Q. Stateful E scores commits; code E, Q score isolated candidates. Address and encoding details appear in the appendix.

5.2 Main comparison: equivalent edits through different interfaces

We compare delivery interfaces on *identical supplied edits*, not autonomous repair agents. The QuixBugs set contains all 40 programs; 40 HumanEvalFix-Python tasks were hash-selected before inference from 133 eligible localized repairs (Appendix J). Every path sees the buggy file, correct replacement text, and source address; only the assembled canonical file is withheld. *Oracle Apply* directly installs the supplied edit without a model. *Ours* asks the model to return a local operation;

Full Rewrite asks it to return the file. Other rows are strict Unified Diff (a format control), Agentless repair-stage `SEARCH/REPLACE` (Xia et al., 2025), one SWE-agent ACI edit (Yang et al., 2024), and CODESTRUCT's auxiliary string-replace action (Kim et al., 2026). None reproduces a complete published agent; untuned AdaEdit grammar prompts (Cheng et al., 2026) appear only in Appendix I.

Model paths use one pinned Qwen2.5-Coder-32B-Instruct checkpoint (Hui et al., 2024) at BF16, greedy decoding, and one attempt per task. Table 1 reports A, Q, E over 40 tasks per path, plus median output tokens and *model-only* GPU generation time; it is not end-to-end wall time. The 4-bit scale study is separate (Appendix I).

Table 1: Delivery of supplied exact edits on two 40-task sets. Oracle Apply directly installs the given spans and values without an LLM; other rows are one-call interfaces using the same 32B BF16 model, not complete published agents. $A/Q/E$: applicable candidate, released-test pass, byte-exact file; Tok/Gen: median output tokens/model-only GPU seconds. Differences are paired percentage points with descriptive 95% task-bootstrap intervals.

	QuixBugs					**HumanEvalFix-Python**				
Method	A	Q	E	Tok	Gen	A	Q	E	Tok	Gen
Oracle Apply	40	40	40	–	–	40	40	40	–	–
Full Rewrite	40	39	24	211.5	7.50	40	39	26	178.0	6.15
Unified Diff	31	31	31	49.5	1.94	38	38	37	68.0	2.52
Agentless S/R	40	38	36	40.5	1.61	38	38	37	49.5	1.91
SWE-agent edit	39	38	37	**34.5**	**1.45**	39	39	39	**38.0**	**1.56**
CODESTRUCT string	38	36	36	87.5	3.21	40	33	33	96.0	3.45
Ours	40	**40**	**40**	154.5	5.44	40	**40**	**40**	160.5	5.59
ΔE *vs. Rewrite*	+40.0 [25.0,55.0]					+35.0 [20.0,50.0]				
ΔE *vs. SWE-agent*	+7.5 [0.0,17.5]					+2.5 [0.0,7.5]				
ΔQ *vs. Rewrite*	+2.5 [0.0,7.5]					+2.5 [0.0,7.5]				

Oracle Apply attains 40/40 without inference, showing that this setup measures model-mediated delivery rather than repair discovery. Patch also yields 40/40 applicable, test-passing, byte-exact files on both sets. Rewrite passes 39/40 tests but yields only 24/40 and 26/40 canonical files; its Python AST matches 39/40 and 38/40 targets. Of its 15 and 13 test-passing noncanonical files, 14 and 12 differ only by one leading blank line, respectively; the others change a trailing newline or docstring quotation. Thus the large E gap is predominantly lexical, with only a 1/40 released-test gap per set. The SWE-agent edit action reaches 37/40 and 39/40 exact files with far fewer output tokens and shorter model generation time than Ours; the paired exact-effect intervals include zero. A post-hoc frame audit further separates preservation from replacement correctness (Appendix K).

Table 2: All 133 eligible HumanEvalFix-Python tasks with pinned 32B weights; Unified Diff and the value-only ablation were scheduled separately. A: applicable; Q: released-test pass; E: byte-exact target; AST: structural agreement; $Q - E$: test-passing noncanonical files. Tok/Gen are median output tokens/model-only GPU seconds. Eval is median trial wall time, estimated from consecutive start times for original paths (last trial excluded) and directly logged for the value-only path; it includes test evaluation, not setup or deployment. Every path receives the correct replacement and address; named tools are single actions.

Delivery path	A	Q	E	AST	$Q - E$	Tok	Gen	Eval
Oracle Apply	133	133	133	133	0	–	–	–
Full Rewrite	133	132	81	130	51	187	6.23	6.28
Unified Diff	128	126	124	126	2	66	2.50	2.59
Agentless S/R	130	130	128	130	2	50	1.87	1.96
SWE-agent edit	131	131	131	131	0	38	1.54	1.71
CODESTRUCT string	133	120	119	119	1	99	3.41	3.53
Ours (value-only)	132	132	132	132	0	**28**	**1.17**	**1.23**
Ours	133	**133**	**133**	**133**	0	160	5.43	5.48

The extension covers all 133 eligible HumanEvalFix-Python tasks, rather than the hash-selected 40, in a separate run under the same pinned checkpoint. Patch and direct application produce 133/133 canonical files, versus 81/133 for Rewrite: a paired $+39.1$-point byte difference (descriptive 95% task-bootstrap interval $[30.8, 47.4]$). Released tests differ by only $1/133$ ($+0.8$ points, interval $[0, 2.3]$); 47 of Rewrite's 51 test-passing noncanonical files omit one leading newline. Rewrite matches the target AST on 130/133 tasks. The SWE-agent edit path is 131/133 exact with much lower output cost than full-envelope Patch. Keeping the descriptor at runtime and returning only values gives 132/133 exact, with medians of 28 output tokens and 1.17 seconds of model generation; its one loss is an invalid JSON response. This changes encoding rather than scope and nearly closes the cost gap, without a detected exact-rate advantage over SWE-agent (paired discordance $2 : 1$; Appendix J.1). The 7B ledger has Rewrite ahead by one released-test pass (37/40 versus 36/40); its counts appear in Appendix Table 21.

On real multi-operation subsets, local versus full delivery passes released tests on 12/12 versus 11/12 HumanEvalFix cases and public examples on 40/40 versus 36/40 CodeEditorBench cases; canonical-file counts are 12/12 versus 0/12 and 40/40 versus 21/40, respectively (Appendix M). The latter is a derived public-example study, not an official CodeEditorBench score.

5.3 Generative scope under different task supports

Historical supplied-intent binary trials yield 240/240 Patch versus 142/240 Rewrite exact commits under one DeepSeek configuration, and 99/100 versus 61/100 under a separate MiniMax configuration. Restricting DeepSeek to the two lengths shared with MiniMax reduces its descriptive gap from 40.8 to 11.7 points. On two aligned 48-task deployments, the Patch–Rewrite difference changes from $+41.7$ to $+2.1$ points; a model alias does not identify immutable weights or client conditions (Appendix O). Thus hosted effects are configuration-specific, not a portable reliability constant.

5.4 Separating scope, address, and encoding

The same-envelope Qwen probe changes only requested output width, expanding Γ_d beyond the single writable bit: exact commits are 40/40, 1/40, 4/40, and 6/40 for widths 1, 8, 32, and 145. The drop beyond width 1 is sharp, but success rises among the three wider widths, ruling out a monotone span law. A post-hoc audit finds wrong-location source matches and omitted edits (Appendix F).

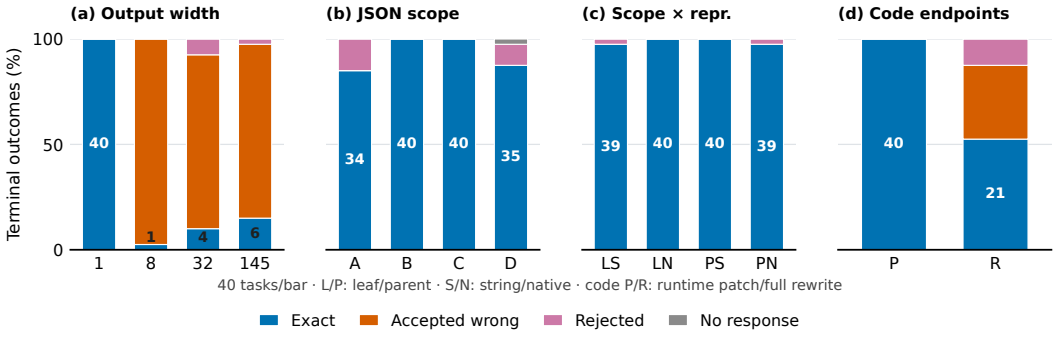


Figure 2: Four compact diagnostics from retained outcomes ($n = 40$ per bar; counts are printed on exact segments). Colors partition unconditional exact, accepted-wrong, rejected, and no-response outcomes. (a) Binary returned width in bits; (b) JSON arms A (leaf), B (resolved parent), C (unresolved parent), D (full root); (c) fresh-intent leaf/parent (L/P) by string/native representation (S/N); (d) historical Qwen3.8 QuixBugs local Patch (P) versus full Rewrite (R), where 35/40 rewrites pass tests although only 21/40 are byte-exact. Panels use separate evaluation setups and are not a pooled comparison.

The public-JSON A–D study gives 34/40, 40/40, 40/40, and 35/40 exact commits, with no prespecified address/scope contrast passing Holm correction. The fresh-intent scope-by-representation factorial is near ceiling (39/40, 40/40, 40/40, 39/40), with no detected contrast. Its new intents do not rerun

the original six A failures, which were malformed nested scalar strings; neither near-ceiling result establishes that scope or representation never matters (Appendix G).

The ablations probe different obligations: width changes scope within one envelope, A–D changes reconstruction and address location, and the factorial changes nested serialization. Failures can omit the intended edit (T), encode a scalar incorrectly, or reconstruct too much source. Reducing Γ_d removes a preservation obligation on U, but the width probe also has T failures. Thus no single response-length mechanism explains all arms.

5.5 Generative exposure: payload sensitivity and failure anatomy

Figure 3 plots two separately frozen studies that move an unchanged payload into or out of Γ_d. Within each provider and dose, leaf and parent share the full root, supplied intent, resolved address, evaluator, and outer request budget. In the opaque-string stress test, MiniMax leaf success changes from 38/40 to 40/40 across 512 to 24,576 characters, while parent success falls from 36/40 to 19/40. The run-frozen scope-by-volume interaction is $+47.5$ points (paired 95% CI $[32.5, 62.5]$; 19 positive and zero negative task interactions; $p = 3.8 \times 10^{-6}$). Qwen points in the same direction: at high volume it is 40/40 leaf versus 29/40 parent, with a descriptive $+25.0$-point interaction. The between-provider difference is leave-one-task-out fragile (Appendix G.5).

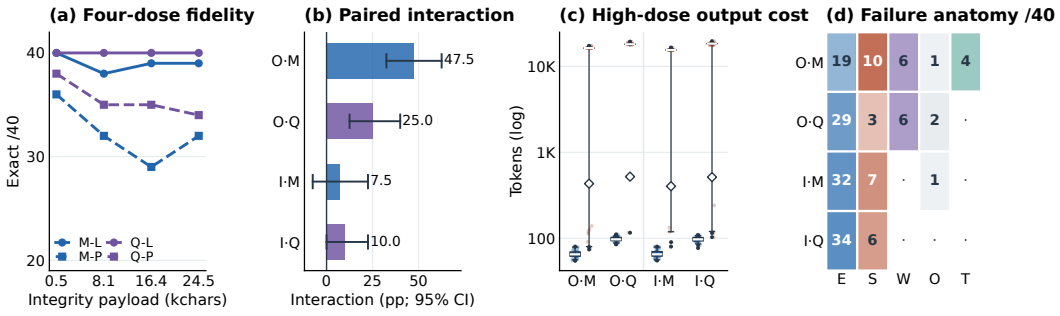


Figure 3: Four diagnostics of two frozen volume studies ($n = 40$ paired tasks/arm). (a) Integrity-list exact commits across dose. (b) Task-paired scope-by-dose effects with bootstrap 95% CIs; only opaque MiniMax was a prespecified primary test. (c) High-dose output-token distributions; diamonds mark low-dose parent medians, and four timeouts lack usage. (d) High-dose parent terminal counts: E exact, S schema rejection, W accepted-wrong, O envelope/type error, T timeout. O/I denote opaque/integrity, M/Q MiniMax/Qwen, and L/P leaf/parent; substrates and hosted snapshots differ.

The four-dose transfer aggregates verbatim lockfile integrity values into JSON lists. MiniMax parent exact counts are 36, 32, 29, and 32/40, versus leaf counts 40, 38, 39, and 39/40. Its first run-frozen gate is nonsignificant ($+10.0$ points, CI $[-2.5, 22.5]$, $p = .145$); Qwen's nominal slope is also nonsignificant. Endpoint interactions are only $+7.5$ and $+10.0$ points, with descriptive paired intervals including zero. The task-level flow (Appendix Figure 5) exposes turnover hidden by these aggregates: in opaque MiniMax parent delivery, 18 tasks lose exactness between doses, one gains it, and 18 remain exact. In the list study, parent median completion cost rises $38.5\times$ (403 to 15,502 tokens) for MiniMax and $35.7\times$ (515 to 18,381) for Qwen, while leaf medians stay around 65–99; the per-call boxes expose early stops. Cost separation is clearer than a monotone reliability slope. The failure matrix adds a distinct stage view: each opaque provider has six accepted-but-wrong high-dose parents, whereas the list arms have none; list failures are almost entirely schema/type rejections (8/8 MiniMax, 6/6 Qwen). This is descriptive across different payloads and hosted snapshots, not a causal substrate comparison; payload forensics and a relaxed-budget diagnostic are in Appendix G.5–G.7.

5.6 Content structure at matched token and character lengths

We cross three unchanged payload types (periodic, prose, and high-entropy) with leaf/parent delivery on the same 40 public-JSON tasks. The original factorial fixes 384 or 2,048 tokenizer tokens; an exploratory replication fixes 2,000 characters instead. Within each study, paired arms share the

intended edit and full input. Figure 4 aligns exactness, failure stages, and the cost of reproducing protected content.

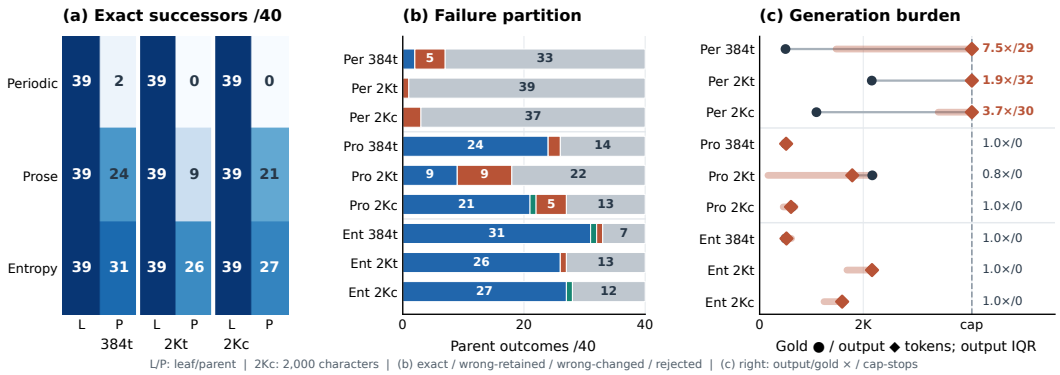


Figure 4: Content structure in 480 token-matched and 240 exploratory character-matched calls ($n = 40$/cell). (a) Exact successors for leaf (L) and parent (P). (b) Unconditional parent outcomes: exact, wrong with protected content retained/changed, and rejected. (c) Parent gold versus observed output tokens with observed IQR; right labels give output/gold ratio and cap stops. Every gold response fits the 4,096-token cap; 2Kc fixes characters, not tokens.

Leaf remains 39/40 exact in every token-matched cell. As dose grows, parent falls from 2 to 0/40 on periodic text, 24 to 9/40 on prose, and 31 to 26/40 on high entropy. At 2,048 tokens, 33/40 periodic outputs overrepeat and 32/40 stop at the cap; prose has no cap stops. The separate 2,000-character study yields 0/40 periodic, 21/40 prose, and 27/40 high entropy parent exactness, while each leaf class stays at 39/40. Periodic parents are rejected 37/40 times, versus 13/40 and 12/40; despite fewer tokens than entropy, 30/40 outputs stop at the cap and 33/40 overrepeat. Length alone cannot explain the contrast, but tokenization and lexical form remain coupled (Appendix N.4–N.6).

5.7 Failure boundaries outside generative exposure

On 40 public-configuration edits, native leaf reaches 26/40 exact states versus 40/40 for parent and full; all 14 errors double-encode a string, and an exploratory clarification recovers 40/40. On source-disjoint longer states, full gold cannot fit a 256-token cap, yet at 4,096 tokens full reaches 38/40 versus leaf's 34/40, again due to scalar encoding. Smaller scope is not universally better (Appendix N.1, N.7).

With nonwritable decoys, source-bound addressing retains 39/40 exact files; naive first-match selects the wrong location on all 40. In a separate inferred-intent pipeline, local versus full-file delivery differs in byte equality (23/40 versus 0/40) but ties at 32/40 released-test passes. Thus the measured gain remains conditional lexical fidelity, not an established behavior gain (Appendix N.2, N.3). Five paired candidate-output case studies (Appendix D) show where our source-bound action succeeds while comparison actions fail to apply or alter the candidate; they illustrate mechanisms rather than their frequency.

6 Conclusion

Generative scope measures the source content a model must reproduce to deliver an update. With correct intent and binding, local composition preserves untouched state by construction; direct application needs no model. Fixed-intent code results show large canonical-byte but small released-test differences against rewriting, driven mainly by lexical changes. A single SWE-agent edit action approaches Patch fidelity at lower output cost. Encoding, address ambiguity, and output limits can reverse the scope ranking; inferred-intent tests tie, and a later hosted rerun attenuates the gap. Scope provides a preservation invariant and measurement axis, while end-to-end benefit remains open.

REFERENCES

Anthropic. Building effective agents. Engineering at Anthropic, December 2024. URL https://www.anthropic.com/engineering/building-effective-agents.

Zhaoyu Bai and Jiaqi Cai. PatchOptic for Shared-State LLM Workflows with Projected Views and Verified Structured Updates. *arXiv preprint arXiv:2607.05483*, 2026. URL https://arxiv.org/abs/2607.05483.

Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic Transactions for Tool-Using LLM Agents. *arXiv preprint arXiv:2606.17573*, 2026. URL https://arxiv.org/abs/2606.17573.

Wei Cheng, Yongchang Cao, Chen Shen, Binhua Li, Jue Chen, Yongbin Li, and Wei Hu. To diff or not to diff? structure-aware and adaptive output formats for efficient LLM-based code editing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 29677–29694. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1483. URL https://aclanthology.org/2026.findings-acl.1483/.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Evgeniy Glukhov, Michele Conti, Egor Bogomolov, Yaroslav Golubev, and Alexander Bezzubov. Diff-XYZ: A benchmark for evaluating diff understanding. *arXiv preprint arXiv:2510.12487*, 2025. URL https://arxiv.org/abs/2510.12487.

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, Shuyue Guo, Xingwei Qu, Xiang Yue, Ge Zhang, Wenhu Chen, and Jie Fu. CodeEditorBench: Evaluating code editing capability of large language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2404.03543.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder Technical Report. *arXiv preprint arXiv:2409.12186*, 2024. URL https://arxiv.org/abs/2409.12186.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Myeongsoo Kim, Chao-Chun Hsu, Dingmin Wang, Shweta Garg, Varun Kumar, and Murali Krishna Ramanathan. CODESTRUCT: Code agents over structured action spaces. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13290–13306. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.607. URL https://aclanthology.org/2026.acl-long.607/.

Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge. In *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, pp. 55–56, 2017. doi: 10.1145/3135932.3135941. URL https://doi.org/10.1145/3135932.3135941.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. Syntactic and semantic control of large language models

via sequential Monte Carlo. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a2d537e69a6c6638a3630eef835f07de-Paper-Conference.pdf.

Jonathan Mallinson, Aliaksei Severyn, Eric Malmi, and Guillermo Garrido. FELIX: Flexible text editing through tagging and insertion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1244–1255. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.111. URL https://aclanthology.org/2020.findings-emnlp.111/.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. Encode, tag, realize: High-precision text editing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 5054–5065. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1510. URL https://aclanthology.org/D19-1510/.

Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. OctoPack: Instruction tuning code large language models. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/1ec299a5229034141e58aeded0d0b9de-Abstract-Conference.html.

Shaan Nagy, Timothy Zhou, Nadia Polikarpova, and Loris D'Antoni. ChopChop: A programmable framework for semantically constraining the output of language models. *Proceedings of the ACM on Programming Languages*, 10(POPL):1905–1932, 2026. doi: 10.1145/3776708. URL https://doi.org/10.1145/3776708.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Introducing structured outputs in the API. OpenAI website, August 2024. URL https://openai.com/index/introducing-structured-outputs-in-the-api/.

Sheena Panthaplackel, Miltiadis Allamanis, and Marc Brockschmidt. Copy that! editing sequences by copying spans. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13622–13630, 2021. doi: 10.1609/aaai.v35i15.17606. URL https://ojs.aaai.org/index.php/AAAI/article/view/17606.

Felix Stahlberg and Shankar Kumar. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 5147–5159. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.418. URL https://aclanthology.org/2020.emnlp-main.418/.

Chaozheng Wang, Zezhou Yang, Shuzheng Gao, Cuiyun Gao, Zongjie Li, Yichen Li, Ting Peng, Hailiang Huang, Yuetang Deng, and Michael R. Lyu. Cascaded code editing: Large-small model collaboration for effective and efficient code editing. *arXiv preprint arXiv:2604.19201*, 2026. URL https://arxiv.org/abs/2604.19201.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

Chunqiu Steven Xia and Lingming Zhang. Less training, more repairing please: Revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 959–971.

ACM, 2022. doi: 10.1145/3540250.3549101. URL https://doi.org/10.1145/3540250.3549101.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 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, pp. 50528–50652, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

Simon Yu, Derek Chong, Ananjan Nandi, Dilara Soylu, Jiuding Sun, Christopher D. Manning, and Weiyan Shi. Shepherd: Enabling Programmable Meta-Agents via Reversible Agentic Execution Traces. *arXiv preprint arXiv:2605.10913*, 2026. URL https://arxiv.org/abs/2605.10913.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025. URL https://arxiv.org/abs/2512.24601.

Yikai Zhang, Jiaxin Pei, Kenan Li, Qirui Jin, Maoquan Wang, Jin Pan, Yu Kang, Shengyu Fu, Elsie Nallipogu, Junjie Hu, Yufan Huang, and Zijian Jin. SWE-Edit: Rethinking code editing for efficient SWE-Agent. *arXiv preprint arXiv:2604.26102*, 2026. URL https://arxiv.org/abs/2604.26102.

APPENDIX CONTENTS

This guide lists the supplementary analyses in reading order. The theoretical analysis follows the evaluation setup; subsequent sections give controlled experiments, program-repair studies, robustness checks, and interpretive boundaries.

Section	Contents	Page
A AI Use Statement	Disclosure of AI assistance and author responsibility.	13
B Evaluation Setup and Scope	Paired-task setup, outcomes, datasets, and comparison boundaries.	13
C Theoretical Analysis	Frame decomposition, scope guarantees, and paired estimands.	14
D Case Studies	Five traced candidate-output contrasts and their mechanisms.	18
E Binary Fixed-Intent Evidence	Controlled one-bit transition results.	21
F Generative-Scope Probe	Same-envelope comparison of returned-source scope.	22
G Structured JSON Experiment	Hierarchy, representation, payload, and budget analyses.	23
H Candidate Fidelity	Canonical-file fidelity beyond released tests.	29
I Published Edit Interfaces	Same-task adaptations of published delivery formats.	29
J HumanEvalFix-Python Setup	Task eligibility, sampling, and fixed-intent construction.	32
K Source-Frame Audit	Structural preservation and replacement-error diagnosis.	35
L Address and Boundary Studies	Address binding and multi-span failure modes.	36
M Multi-Operation Repair	Delivery results for real multi-operation edits.	37
N Supplementary Experiments	Transfer, ambiguity, content, and budget checks.	38
O Temporal Sensitivity	Replication across deployment snapshots.	43
P Limitations	Assumptions, scope, and interpretation of the evidence.	43
Q Ethics Statement	Data provenance and deployment boundaries.	43

A AI USE STATEMENT

Large Language Models (LLMs) were employed solely to support the linguistic refinement of this manuscript. Their use was limited to improving grammar, clarity, sentence construction, and overall presentation of the text. LLMs were not involved in the scientific or intellectual development of this work, including research design, hypothesis formulation, data analysis, experimental evaluation, or interpretation of the findings. All scientific contributions and conclusions were developed by the authors. The authors have carefully reviewed the final manuscript and assume full responsibility for its content.

B EVALUATION SETUP AND SCOPE

The primary comparisons concern *delivery after a correct edit intent has been supplied*; explicitly labeled inferred-intent results are supplementary. The authoritative input, intended target, and allowed edit footprint are fixed within each paired task. The interface determines which unchanged source content the model must regenerate and which content the runtime inherits directly. A response must yield an applicable candidate and pass the study's acceptance checks before a transition could change authoritative state. Rejection, timeout, malformed output, and missing candidates remain failures in unconditional denominators. Program repair is evaluated in an isolated candidate state without a common persistent write across method paths. We report canonical file equality, official-test pass, and candidate availability separately: a finite test suite can accept a noncanonical file. Hosted-model configurations, local checkpoints, and later deployment snapshots are analyzed within their own paired blocks rather than pooled. The experiments vary generated source scope, representation, and address resolution; several paths change more than one factor.

Comparisons described as prespecified were frozen in local run plans before their respective calls; no public prospective registration is claimed.

The binary tasks are one-bit transitions over $\mathrm{GF}(2)$ with the same source and correct bit edit in paired arms. The 40 public JSON files come from pinned Prettier, ESLint, TypeScript, VS Code,

and Node-RED revisions; two source-disjoint files are smoke tests only. Opaque payloads are 512 or 24,576 characters, and the verbatim npm-lockfile integrity lists realize 491, 8,135, 16,367, or 24,501 characters. The list study retains the opaque study's 40 document/intent blocks with new payload construction. JSON arms A–D return a resolved leaf, pre-cut parent, root-located parent, or complete root; a fresh-intent follow-up crosses leaf/parent with JSON-in-string/native-JSON replacement. The main 32B BF16 method comparison uses a 4,096-token cap, 180-second model deadline, and no repair retry. Its seven original HumanEvalFix paths rotate within task; added formats and two QuixBugs tools were scheduled separately. Reported GPU generation latency excludes input tokenization, parsing, tool execution, and tests.

Table 3: Evaluation map. n is the number of paired task blocks unless repeated calls are specified. Each row is analyzed within its own observed configuration.

Substrate	Blocks / calls	Controlled interface contrast	Main endpoint
Binary successor	DeepSeek 80×3; MiniMax 100	Runtime Patch vs. full Rewrite	Exact commit
Same-envelope binary	40×4	Returned width $1/8/32/145$	Exact commit
Public JSON	40×4, twice	Address/scope A–D; scope × representation	Deep-equal commit
Payload carry-forward	320 opaque; 640 SRI	Leaf vs. parent across provider and dose	Deep-equal commit
Protected content	40×12 token; 40×6 character	Payload class × leaf/parent at matched size	Exact commit, output burden
QuixBugs / HumanEvalFix Python	40 Quix; 40 selected and 133 eligible Human	Oracle Apply; local, full, format/tool actions	Candidate, exact file, tests
Real multi-operation Python	12 HumanEvalFix; 40 CodeEditorBench	Compact local values vs. full Rewrite	Exact file; author/public test

C Theoretical Analysis

C.1 Edit-Context Decomposition

Let $\Omega(S)$ be the source-origin atoms of authoritative state S: bit coordinates, source positions for code, or covered tree locations for JSON. The supplied intent I^* specifies an intended footprint $W \subseteq \Omega(S)$ and a canonical successor S^*. Its descriptor D^* satisfies $\mathrm{Chk}(D^*, S)$. A proposed descriptor D declares a possibly different footprint W_D. The mechanically checkable predicate $\mathrm{Chk}(D, S)$ requires the expected source version and old values to match S, and its addresses to be ordered, in bounds, and nonoverlapping. The separate semantic premise $\mathrm{Align}(D, S, I^*)$ says that those addresses and frame boundaries designate the *intended* footprint W and its unchanged complement. Checking a hash and old values does not establish alignment.

For code, take $S \in \Sigma^*$ and let a checked D specify m spans in the original string. It induces the unique decomposition

$$S = U_0 \| V_1 \| U_1 \| \cdots \| V_m \| U_m, \qquad F_D(S) = (U_0, \ldots, U_m), \tag{8}$$

where V_j are the declared writable values and U_j the ordered untouched segments. A pure insertion has $V_j = \epsilon$ and consumes no source position, but still needs an intended boundary. Coincident insertions require an explicit order and are excluded here. For bits, D names coordinates; for JSON, it names nonoverlapping subtree paths. The frame is the complement projection for bits and JSON and the ordered U_j for code. JSON topology and type constraints are checked by the declared invariant gate, unless explicitly made part of the replacement domain.

C.2 Generative Scope Control

For interface d, let $B_d \subseteq \Omega(S)$ cover the source-origin region whose successor content the model must return. Under alignment, $W \subseteq B_d$ and the unchanged region exposed to regeneration is

$\Gamma_d = B_d \setminus W$; the runtime owns $\Omega(S) \setminus B_d$. Thus $\Omega(S) = W \dot\cup \Gamma_d \dot\cup (\Omega(S) \setminus B_d)$, as in Equation 3. For JSON, a returned parent covers its descendant source atoms; for code, a returned range has explicit source boundaries. A zero-length insertion consumes no original atom but retains its boundary anchor in D; its inserted bytes are generated. Exposure may be measured in source bits, serialized bytes, or structural fields, each within its own substrate. It is not total response tokens: addresses, escaping, values, and syntax contribute separately.

Proposition 2 (Scope containment)**.** *Suppose* $\mathrm{Chk}(D, S)$ *and* $\mathrm{Align}(D, S, I^*)$ *hold,* $W \subseteq B_d$*, and the resolver replaces only the returned region* B_d*, inserts only at source boundaries covered by that region's binding, and carries* $\Omega(S) \setminus B_d$ *from the same authoritative* S*. Then every constructed candidate preserves the source outside* B_d*; any collateral change to an intended-unchanged source atom is confined to* Γ_d*. At local scope* $B_d = W$*, a resolver that performs* put_D *at* D*'s declared boundaries preserves the entire intended frame. A parent return guarantees the frame outside that parent; a full-state return* $B_d = \Omega(S)$ *makes the structural claim vacuous.*

Proof. By the resolver premise, every atom outside B_d is copied from S. Under alignment, each intended-unchanged atom lies either in Γ_d or outside B_d. Consequently, a change to an intended-unchanged source atom can occur only in Γ_d; the insertion premise also keeps inserted content within the returned region. If $B_d = W$, this set is empty; the additional put_D premise ensures that even a zero-length insertion uses the intended boundary. For code, the copy statement uses the ordered unchanged segments outside the returned range and their transported boundaries. For JSON and bits, it is the corresponding coordinate or subtree copy. □

The proposition bounds *where* a constructed candidate can change the frame; it does not order the model's probability of returning a correct edit. All interfaces may read the same full source. A shorter scope can still lose candidates through response syntax, address binding, or an incorrect replacement. No independent-error law, monotone length law, or universal token/latency ranking follows.

C.3 Source-Grounded Composition

If $\mathrm{Chk}(D, S)$ fails, composition returns no candidate. Otherwise, for replacement tuple $z = (z_1, \ldots, z_m)$, the source-bound code action is

$$\mathrm{put}_D(S, z) = U_0 \| z_1 \| U_1 \| \cdots \| z_m \| U_m. \tag{9}$$

For code let $\mathcal{Z}_D = (\Sigma^*)^m$; for bits use values at declared coordinates, and for JSON use well-formed replacement subtrees at declared paths. Define the admissible candidate family and source-relative relation *extensionally*:

$$\mathcal{C}_D(S) = \begin{cases} \{\mathrm{put}_D(S, z) : z \in \mathcal{Z}_D\}, & \mathrm{Chk}(D, S), \\ \varnothing, & \text{otherwise}, \end{cases} \qquad \mathcal{R}_D(S, S') \iff S' \in \mathcal{C}_D(S). \tag{10}$$

Membership for a full-file candidate is a property of its resulting string or tree; it does not presume a uniquely recoverable D'. Repeated code substrings can admit multiple replacement-tuple witnesses, but membership remains well defined. The transported descriptor $D[z]$ below is constructed by a known put_D operation, so its boundaries are not inferred from an arbitrary full-file response. The relation is relative to the *declared* frame; under Align it is the intended preservation relation. Neither $\mathcal{C}_D$ nor $\mathcal{R}_D$ requires the replacement to equal z^*.

Proposition 3 (Frame and round-trip laws)**.** *For every checked* D *and permitted* z, z_1, z_2*, the source-bound operator satisfies*

$$\begin{aligned} &F_{D[z]}(\mathrm{put}_D(S, z)) = F_D(S), && \mathrm{get}_{D[z]}(\mathrm{put}_D(S, z)) = z, \\ &\mathrm{put}_D(S, \mathrm{get}_D(S)) = S, && \mathrm{put}_{D[z_1]}(\mathrm{put}_D(S, z_1), z_2) = \mathrm{put}_D(S, z_2). \end{aligned} \tag{11}$$

Here $D[z]$ *is the transported descriptor for the constructed successor: its boundaries, expected source version, and expected old values refer to* $\mathrm{put}_D(S, z)$*. The operator* get *extracts declared writable spans. The first equality holds even for a wrong replacement value.*

Proof. Equation 9 reuses every U_j in order. The transported boundaries enclose precisely z_j, proving extraction. Substituting the original V_j reproduces S. A second replacement at those transported

boundaries removes z_1 and leaves the same U_j around z_2, proving the final law. Coordinate and subtree replacement give the same componentwise argument. Structural validity is a separate gate condition. □

Proposition 4 (Transactional safety is conditional)**.** *Suppose a transition checks D against authoritative S, constructs $S' = \mathrm{put}_D(S, z)$, and commits S' atomically against that* same source version *only if a sound predicate $\mathcal{V}(S')$ accepts. Every committed successor then preserves the declared frame $F_D(S)$ and satisfies $\mathcal{V}$. If $\mathrm{Align}(D, S, I^*)$ also holds, it preserves the intended frame. A failed check, gate, or version comparison does not install this candidate.*

Proof. The first law in Proposition 3 proves declared-frame preservation; alignment identifies this with the intended frame. Gate soundness proves $\mathcal{V}$ on accepted candidates. Atomic compare-and-commit prevents a partial multi-span write and rejects a source changed after the check. On rejection, this transition makes no write; another concurrent transition may already have changed the authoritative state. □

This is a frame guarantee, not a proof that the model chose the intended z^*. Nor does $\mathcal{V}$ prove arbitrary program semantics: a syntax check guarantees syntax, while finite tests certify only their observed outcomes. The code experiments score isolated candidates, so they do not test this persistent atomic-commit premise. A finite source hash is a practical version check under its collision assumptions; the theorem assumes a sound source-version comparison.

Proposition 5 (Insertion boundaries need an address witness)**.** *Let a string edit insert at intended boundary k with expected old value ϵ. An old-value check alone cannot distinguish k from any other boundary. If a source tag is recomputed after a change while the old numeric offset is retained, that new tag alone does not restore* Align*. Conversely, if fresh intent evidence identifies the current intended boundary by exact adjacent left and right contexts L, R, and $L\|R$ occurs at only one permitted boundary in the current source, an accepted insertion address equals the intended boundary.*

Proof. Every zero-length slice $S[k' : k']$ equals ϵ. For example, insert Z between a and b in $S = ab$ at offset 1. After a prefix x changes the source to xab, rebinding the source tag but retaining offset 1 yields $xZab$, whereas the intended rebased edit yields $xaZb$. The stale descriptor can pass mechanical checks and preserve its *own* declared frame, but fails alignment. For the converse, acceptance at a different boundary would give a second occurrence of the required adjacent contexts, contradicting uniqueness. □

The context condition is sufficient rather than automatic: repeated neighboring text can make a witness ambiguous. An unchanged original source tag rejects an unapproved source version; a deliberate rebase needs fresh, unique location evidence. This distinction explains the stale-offset boundary experiment without treating it as a counterexample to the structural frame law.

A common attribution for outcome diagnosis. For an applicable code candidate S', fix a task-level rule α_{D^*,z^*} before scoring model responses. It partitions S' into alternating putative unchanged and edited strings,

$$S' = \widetilde{U}_0 \| \widetilde{z}_1 \| \widetilde{U}_1 \| \cdots \| \widetilde{z}_m \| \widetilde{U}_m, \quad U = \mathbf{1}[(\widetilde{U}_j)_j = (U_j)_j], \quad T = \mathbf{1}[(\widetilde{z}_j)_j = z^*]. \tag{12}$$

The rule selects the canonical (U, z^*) partition when $S' = S^*$; for any other $S' \in \mathcal{C}_{D^*}(S)$ it selects a frame-preserving witness with a fixed tie-break; elsewhere it selects a fixed partition. Such a rule exists because empty parts are permitted. Consequently $U = 1$ exactly for the image-set relation, and $T = U = 1$ exactly when $S' = S^*$. In particular, an arbitrary witness for a multi-span full rewrite cannot be substituted after seeing the output: even a byte-exact file may have several replacement tuples. Bits and fixed-topology JSON have direct coordinate/path attributions. For ambiguous code outputs, T outside exact success depends on the declared rule; we report E and U for code but do not estimate its separate T factor or the threshold components below.

Exact-success tradeoff. For interface d, let C_d denote response-contract acceptance, A_d an applicable candidate satisfying the mechanical source and address checks, T_d the intended replacement under the common attribution, U_d preservation of the *intended* frame, and V_d the declared gate and, when relevant, successful installation. Set $T_d = U_d = 0$ when no candidate exists. On a nontrivial unique-target task ($S^* \neq S$), with rejection scored as failure, exact success is $E_d = C_d \cap A_d \cap T_d \cap U_d \cap V_d$. In the code comparison, $V_d \equiv 1$ once a candidate passes any path-specific application gate folded into A_d; released tests Q_d are a separate endpoint.

Put $J_d = C_d \cap A_d \cap T_d \cap V_d$, $j_d = \Pr(J_d)$, and $q_d = \Pr(U_d \mid J_d)$ when $j_d > 0$ (set $q_d = 0$ otherwise). The chain rule gives $\Pr(E_d) = j_d q_d$ without independence. On a task distribution where local descriptors satisfy Align whenever J_{local} occurs, Proposition 2 gives $q_{\text{local}} = 1$ if $j_{\text{local}} > 0$. Thus

$$\Pr(E_{\text{local}}) - \Pr(E_{\text{full}}) = j_{\text{local}} - j_{\text{full}} q_{\text{full}} = \Pr(J_{\text{local}} \setminus J_{\text{full}}) - \Pr(J_{\text{full}} \setminus J_{\text{local}}) + \Pr(J_{\text{full}} \cap \neg U_{\text{full}}). \tag{13}$$

The first two paired terms capture differences in non-frame obligations; the third is a full-return frame failure among responses meeting those obligations. If a gate detects every frame error before acceptance, this third term may be zero because the loss is instead counted in J_{full}. The identity is descriptive, not a causal attribution to individual factors.

Proposition 6 (Paired threshold for a reliability gain)**.** *Under the aligned local-splice premises above and a joint distribution of paired responses on the same tasks, define $\kappa = \Pr(J_{\text{local}} \setminus J_{\text{full}})$, $\lambda = \Pr(J_{\text{full}} \setminus J_{\text{local}})$, and $\rho = \Pr(J_{\text{full}} \cap \neg U_{\text{full}})$. Then local delivery has strictly higher exact-success probability if and only if $\kappa + \rho > \lambda$. In particular, $\rho > \lambda$ is sufficient even when the local path loses some non-frame obligations.*

Proof. Partition J_{local} and J_{full} into their shared and exclusive events. The shared mass cancels in $j_{\text{local}} - j_{\text{full}}$, while $j_{\text{full}}(1 - q_{\text{full}}) = \Pr(J_{\text{full}} \cap \neg U_{\text{full}})$. Equation 13 becomes $\kappa - \lambda + \rho$, yielding both claims. □

This threshold is a bookkeeping consequence of the frame law, not a learned-model performance theorem. Its terms need separately identified T/U outcomes to estimate; the primary code table reports $A/Q/E$ instead. A compact local response may have a difficult schema or address format, so λ can exceed the frame failures it avoids. The same-envelope width probe is nonmonotone among its three wider conditions; no independent per-character error law is assumed.

There is no unconditional dominance. For $S = (0, 0)$, $W = \{1\}$, and target $S^* = (1, 0)$, a malformed local response paired with an exact full response makes only full rewriting succeed. A valid local value 1 paired with full response $(1, 1)$ makes only local splicing succeed. Both constructions have the same correct intent and satisfy the structural theorem.

C.4 Outcome decomposition and paired inference

For the same event definitions, exact success obeys the full chain rule

$$\Pr(E) = \Pr(C) \Pr(A \mid C) \Pr(T \mid C, A) \Pr(U \mid C, A, T) \Pr(V \mid C, A, T, U). \tag{14}$$

This chain-rule account entails no independence or causal claim about an individual factor. When a study records only composite acceptance, its factors are not separately estimated. The program tables operationalize E as byte equality of an applicable candidate to the canonical successor, without claiming to measure a persistent commit. A finite official-test pass Q does not replace target equality.

For paired tasks $i = 1, \ldots, n$, let $Y_i(a)$ and $Y_i(b)$ be unconditional exact-success indicators and $D_i = Y_i(a) - Y_i(b)$. The task-average difference and its task-level standard error are

$$\widehat{\Delta} = \frac{1}{n} \sum_i D_i, \qquad \widehat{\mathrm{SE}}(\widehat{\Delta}) = \sqrt{\frac{1}{n(n-1)} \sum_i (D_i - \widehat{\Delta})^2}. \tag{15}$$

A population standard-error interpretation requires independently sampled task blocks; repeated calls or doses within a task remain clustered. For binary endpoints, if n_{10} tasks favor a alone and n_{01} favor b alone, then $\widehat{\Delta} = (n_{10} - n_{01})/n$. Conditional on $m = n_{10} + n_{01}$ discordances, the two-sided exact sign test under exchangeability has probability $\min\{1, 2 \Pr[\mathrm{Binomial}(m, 1/2) \leq \min(n_{10}, n_{01})]\}$. Such inference estimates a contrast within the specified task and deployment distribution; the frame laws alone do not identify its sign.

D CASE STUDIES: TRACED DELIVERY DECISIONS

These five examples are selected from the same one-call, 32B BF16 QuixBugs40 and HumanEvalFix-Python40 runs as Table 1. Every compared path received the full authoritative file and the same supplied correct edit intent; the canonical successor was withheld. The QuixBugs CODESTRUCT tool extension was scheduled separately on the same frozen tasks and checkpoint. *Ours (runtime Patch)* is the source-bound edit interface evaluated as *Ours* in Table 1; "Patch" names its returned action, not a separate baseline. The compared output is the *candidate file after applying* each returned action, except where no candidate could be constructed. Code blocks contain local source, target, response, or candidate excerpts as labeled; ellipses and stated whitespace markers are annotations. A displayed abbreviated hash denotes the full source digest carried by the recorded response. In each outcome row, A is an applicable candidate, Q is released-test pass, and E is canonical byte equality. A test-passing, noncanonical file is a state-fidelity failure, not an observed behavioral failure. The CODESTRUCT row denotes its auxiliary one-action string tool, and Agentless denotes its repair-stage `SEARCH/REPLACE` path; neither card represents an end-to-end autonomous agent run.

Case D.1 Repeated old text: `get_factors` (QuixBugs)

Frozen input and target. The source contains two matches for the searched four-space substring `return []`: one inside the eight-space base case and one at the four-space final return. Only the latter is the supplied writable span $[166, 180)$; the base case must survive.

```
S:  if n == 1:                  S*: if n == 1:
        return []                       return []
    ...                             ...
    return []                       return [n]
```

Returned action and actual output. Ours returned the source digest `7eff7ef2...3290d`, the declared span, `old_text=" return []\n"`, and `new_text=" return [n]\n"`. Its candidate file contains both the preserved base case and changed final return:

```
Ours candidate (two separate locations):
    if n == 1:
        return []
    ...
    return [n]
```

The CODESTRUCT auxiliary tool instead returned:

```
str_replace(old_str="    return []\n", new_str="    return [n]\n")
```

Its applier reported `nonunique_old_str`: *no CODESTRUCT candidate file exists* for comparison, because the searched substring occurs twice. Ours uses the correct old value *and* its source-bound address; the runtime carries both untouched regions through composition.

Delivery path	A	Q	E	Candidate state
Ours (runtime Patch)	1	1	1	Final `return [n]`; base case retained.
CODESTRUCT string tool	0	0	0	Ambiguous old string; no candidate.

Method connection. *Edit-Context Decomposition* records which occurrence is editable, while *Source-Grounded Composition* reuses the rest of S. This is an applicability contrast between the recorded one-action paths, not a claim that a longer string search could never disambiguate the file.

Case D.2 A zero-length insertion: `wrap` (QuixBugs)

Frozen input and target. The supplied operation inserts `lines.append(text)` at character boundary $[225, 225)$, immediately before the existing return. Its expected old value is empty, so the boundary is the address witness.

```
S:  lines.append(line)            S*: lines.append(line)

    return lines                      lines.append(text)
                                      return lines
```

Returned actions and actual output. Ours returned the source digest `0dc3363d...1586f`, `start_char=end_char=225`, `old_text=""`, and `new_text=" lines.append(text)\n"`. These excerpts come from the constructed candidate files:

```
Ours candidate (function end):
    lines.append(line)

    lines.append(text)
    return lines
Full Rewrite candidate (function end):
    lines.append(line)

    lines.append(text)

"""
Agentless candidate (docstring, outside edit span):
Wrap Text
    lines.append(text)
Given a long string...
```

The Rewrite candidate omits the existing return and fails tests. Agentless submitted an empty `SEARCH` region:

```
<<<<<<< SEARCH

=======
    lines.append(text)
>>>>>>> REPLACE
```

The applied candidate contains the desired insertion plus five extra copies at other blank-line matches; the docstring excerpt shows one such copy. It passes the released tests but is not the canonical file.

Delivery path	A	Q	E	Candidate state
Ours (runtime Patch)	1	1	1	One insertion; existing return retained.
Full Rewrite	1	0	0	Return statement omitted.
Agentless search/replace	1	1	0	Extra insertions at blank-line matches.

Method connection. The zero-length old value cannot locate an insertion. *Edit-Context Decomposition* supplies the boundary and source identity, and *Source-Grounded Composition* applies one splice while carrying forward the original return and all surrounding text.

Case D.3 Substring replacement doubles indentation: `Python/40`

Frozen input and target. The intended HumanEvalFix edit replaces the entire line at $[484, 515)$, correcting the starting index without changing indentation or the nested loops.

```
S:     for i in range(1, len(l)):
S*:    for i in range(len(l)):
```

Returned actions and actual output. Ours returned the source digest `f976fe5f...02f` and the full-line old/new strings, including their four leading spaces. The CODESTRUCT auxiliary tool instead returned the following substring action:

```
old_str = "for i in range(1, len(l)):\n"
new_str = "    for i in range(len(l)):\n"
```

The source's four leading spaces remain outside `old_str`; replacing only the matched substring adds another four. The following `[4 spaces]` and `[8 spaces]` markers visualize the *actual leading whitespace* in the candidate lines; they are not literal source characters:

```
Ours candidate:       [4 spaces]for i in range(len(l)):
CODESTRUCT candidate: [8 spaces]for i in range(len(l)):
```

CODESTRUCT's candidate fails the released tests.

Delivery path	A	Q	E	Candidate state
Ours (runtime Patch)	1	1	1	Correct loop; four leading spaces.
CODESTRUCT string tool	1	0	0	Same loop; eight leading spaces.

Method connection. *Source-Grounded Composition* replaces the exact declared span once. A string action can be applicable yet miscompose a correct-looking replacement when its searched substring and returned text disagree about ownership of surrounding whitespace.

Case D.4 The edit is present but one word is lost: `Python/113`

Frozen input and target. The supplied operation changes one full line in `odd_count`, at $[665, 803)$. These literal suffixes show the difference; n is inserted into the generated message at runtime.

```
S:  ... +" of "+ str(n) +" the "+ str(n) +"nput."
S*: ... +" of the "+ str(n) +"nput."
```

Returned actions and actual output. Ours returned the source digest `240f7f4d...14b12e` with the exact old and new full-line strings, then composed the canonical file. The following suffixes are taken from the *three candidate files*; the labels and ellipses are annotations:

```
Ours candidate:       ... +" of the "+ str(n) +"nput."
Full Rewrite candidate: ... +" of "+ str(n) +"nput."
CODESTRUCT candidate:  ... +" of "+ str(n) +"nput."
```

This removes `the ` as well as the intended redundant expression. Rewrite additionally drops the initial source newline. Both candidates fail the released tests; neither failure can be explained as harmless leading-whitespace drift.

Delivery path	A	Q	E	Candidate state
Ours (runtime Patch)	1	1	1	Intended message and source frame.
Full Rewrite	1	0	0	Wrong message; initial newline omitted.
CODESTRUCT string tool	1	0	0	Wrong message; source frame otherwise retained.

Method connection. A candidate may pass format and source-application checks ($A = 1$) while changing the intended replacement incorrectly ($T = 0$). The case separates edit-content accuracy from the preservation guarantee and shows why the endpoint chain in Equation 14 contains both factors.

Case D.5 Tests pass while an untouched example changes: `Python/158`

Frozen input and target. The supplied edit changes only the final return expression at $[465, 493)$. The docstring example below is outside the writable span; its unusual double quote is part of the authoritative file and remains in the canonical target.

```
S docstring = S* docstring: ... == ""aaaaaaa"
S return:  return sorted(words)[0]
S* return: return sorted(words, key = lambda x: (-len(set(x)), x))[0]
```

Returned actions and actual output. Ours returned the source digest `f5f402e5...68c3` and the source-bound full-line replacement. These candidate-file excerpts expose the difference *outside* the supplied edit span:

```
Ours candidate (docstring):
    find_max(["aaaaaaa", "bb" ,"cc"]) == ""aaaaaaa"
Full Rewrite candidate (docstring):
    find_max(["aaaaaaa", "bb" ,"cc"]) == "aaaaaaa"
Both candidates (edited return):
    return sorted(words, key = lambda x: (-len(set(x)), x))[0]
```

The rewritten file passes released tests but fails canonical equality because it deletes one quote in the untouched docstring.

Delivery path	A	Q	E	Candidate state
Ours (runtime Patch)	1	1	1	Correct return; original docstring retained.
Full Rewrite	1	1	0	Correct return; one docstring quote deleted.

Method connection. *Generative Scope Control* assigns only the intended return line as successor content to the model and leaves the docstring to the runtime. The example isolates lexical state fidelity: $Q = 1$ supplies no evidence that the docstring was preserved, while the local splice enforces the frame relation under the valid binding.

E Binary Fixed-Intent Evidence

Table 4: Binary exact commits. Differences are Patch minus Rewrite; all failures remain in the denominator. The common-support row is descriptive.

Configuration and support	Patch	Rewrite	Difference	95% CI
DeepSeek, $N \in \{96, 145, 240, 384\}$	240/240	142/240	$+40.8$ pp	$[31.3, 50.4]$
DeepSeek, $N \in \{96, 145\}$	120/120	106/120	$+11.7$ pp	post hoc
MiniMax, $N \in \{96, 145\}$	99/100	61/100	$+38.0$ pp	$[28.0, 48.0]$

E.1 DeepSeek coverage, inference, and failure classes

The DeepSeek matrix has all 1,920 expected rows. The supplied-correct-intent contrast contains 80 task instances across four lengths and three repeats. Patch succeeds on 240/240 and Rewrite on 142/240. After averaging repeats within instance, the effect is $+0.4083$, 95% CI $[+0.3125, +0.5042]$. A 200,000-draw paired sign-flip test with add-one correction yields $p < 5 \times 10^{-6}$. There are 41 Patch-positive instances, none Rewrite-positive, and 39 ties.

The inferred-intent conditions each contain 160 instances times three repeat/day blocks, explaining the $n = 480$ labels in historical tables. Patch succeeds on 228/480 and Rewrite on 124/480; the task-clustered difference is $+0.2167$, CI $[+0.1646, +0.2688]$. This comparison includes intent inference and is not the supplied-correct-intent estimand.

Table 5: DeepSeek terminal outcomes. Internal historical arm codes are expanded here.

Condition	Success	Timeout	Invalid format	Returned wrong	Total
Inferred intent, Patch	228	246	0	6	480
Inferred intent, Rewrite	124	333	9	14	480
Fixed intent, Patch	240	0	0	0	240
Fixed intent, Rewrite	142	31	15	52	240

The supplied-intent Patch arm is 60/60 at each length. Rewrite is 58/60 at $N = 96$, 48/60 at $N = 145$, and falls to 7/60 at $N = 384$. Hence the common-support calculation is 120/120 versus 106/120, $+11.7$ points. It is post hoc and does not pool with MiniMax.

Among the 52 DeepSeek returned-wrong rewrites, Hamming distance to target is one for 48, two for three, and 23 for one. Only four modify coordinates outside the target. The dominant error is edit omission, but format and timeout outcomes remain part of the compound interface effect.

E.2 MiniMax paired analysis

The MiniMax design contains 100 fresh pairs, 50 each at $N = 96$ and 145. Pair order is randomized and balanced, with one attempt per prompt. All 200 outcomes enter the analysis; one Rewrite reaches the 300-second limit and remains a failure.

Patch succeeds on 99/100 and Rewrite on 61/100. The paired table contains 39 Patch-only, one Rewrite-only, 60 both-success, and zero both-failure tasks. The risk difference is $+0.380$, paired-bootstrap CI $[+0.280, +0.480]$, with exact two-sided $p = 7.5 \times 10^{-11}$. At $N = 96$, the rates are 50/50 and 34/50; at $N = 145$, 49/50 and 27/50.

F Same-Envelope Generative-Scope Probe

F.1 Experimental outcomes

The Qwen3.8 probe uses 40 new $N = 145$ instances and four widths in a balanced Latin square. All 160 scheduled attempts enter the analysis, with no retry, timeout, or transport failure. It varies the unchanged source coordinates included in model output under the same response envelope and fixed edit intent; it does not vary input length or establish a general length law.

Table 6: Same-envelope scope-probe terminal outcomes.

Width	Exact	Accepted wrong	Invalid contract	Timeout/transport	Total
1	40	0	0	0	40
8	1	39	0	0	40
32	4	33	3	0	40
145	6	33	1	0	40

The paired width-1 minus width-145 effect is $+0.850$, 100,000-draw task bootstrap CI $[+0.725, +0.950]$. Thinking is disabled and reported reasoning tokens are zero; median completion tokens are 75, 84, 108, and 219 as width increases. Median latency is 4.88, 5.04, 5.34, and 6.48 seconds.

F.2 Exploratory origin analysis

The response-origin audit is post hoc and does not establish a cognitive mechanism. At width 8, 26 of 39 accepted wrong replacements occur as full substrings of authoritative input and have at least one occurrence outside the declared window; only three have an adjacent (± 1) occurrence, so there is

no uniform 0-/1-based explanation. At width 32, 17 of 33 are full input substrings and 15 occur at a wrong location; offsets vary widely and none is adjacent (± 1). Coincidental repeats are retained with every occurrence rather than assigned to a preferred source. At width 145, 16 of 33 accepted wrong candidates exactly equal the authoritative input; all 33 miss the edit, 16 have zero collateral changes, and the remaining 17 have one extra wrong bit.

Table 7: Post-hoc failure-mechanism audit of the same-envelope scope probe. Hamming and collateral means use contract-accepted rows, with exact rows contributing zero.

Width	Exact	Target correct	Mean Hamming	Mean collateral	Diagnostic
1	40/40	40/40	0.00	0.00	delta-only; no unchanged bits
8	1/40	18/40	3.63	3.08	26/39 wrong rows match a source substring outside scope
32	4/40	21/40	13.11	12.68	17/33 source substrings; 15 from a wrong location
145	6/40	6/40	1.28	0.44	16/33 copy input unchanged; all 33 omit the edit

G Structured JSON Experiment

G.1 Public sources and task construction

Forty public JSON documents from Prettier, ESLint, TypeScript, VS Code, and Node-RED form the formal task set; two separate documents served only for feasibility checks. The study uses public project configuration and no private account data. Each formal task contributes one scalar edit at depth at least two.

Credential-, identity-, hash-, and workflow-reference keys are excluded. Replacement is type-preserving: booleans flip, numbers increment, numeric strings increment, semantic versions advance one patch, and other bounded strings receive a deterministic suffix. Each target differs at exactly that leaf.

The frozen invariant requires the same root kind, object-key sets, array lengths, and scalar types at every path. It additionally preserves root package name/version and mapping types, Node-RED node identity/order, and schema identity fields when present. This is a structural/domain invariant, not evidence that every synthetic edit is useful to the upstream software.

G.2 Arms and endpoints

Table 8: Four output-scope and address arms. All use identical envelope keys and receive the full root and correct intent.

Arm	Requested scope	Additional information / runtime action
A: leaf runtime-owned	resolved scalar leaf	Old leaf supplied; model returns nested JSON for replacement scalar; runtime carries every other path.
B: resolved parent	parent subtree	Pre-cut authoritative parent and relative edit path supplied; runtime splices returned successor parent into root.
C: unresolved parent	same parent subtree	No pre-cut parent; model locates it from root using absolute path; runtime splice is otherwise identical to B.
D: full root	root	Model returns successor root; runtime validates and commits it.

The response carries a source-state tag that is checked against the authoritative scope before commit. The primary endpoint is deep equality after parse and commit: whitespace and object-key order are ignored, whereas array order is preserved. Invalid envelope, nested replacement, scope, type, or structural invariant, as well as timeout or transport failure, counts as failure. The 160 formal attempts followed the fixed task and treatment schedule.

G.3 COMPLETE OUTCOMES AND INFERENCE

Table 9: Unconditional structured-state endpoints. "Accepted/gated" is the historical composite of outer keys, usable nested replacement JSON, scope/hash/type checks, and invariant acceptance; it is not the isolated C event in Equation 7. "Parsed" precedes the invariant gate and can exceed accepted/gated.

Arm	Exact	Accepted/gated	Parsed	Invariant	Leaf correct	Candidate
A: leaf runtime-owned	34/40	34/40	34/40	34/40	34/40	34/40
B: resolved parent	40/40	40/40	40/40	40/40	40/40	40/40
C: unresolved parent	40/40	40/40	40/40	40/40	40/40	40/40
D: full root	35/40	35/40	36/40	35/40	35/40	36/40

Table 10: Three run-frozen paired contrasts. Simultaneous intervals are Bonferroni 98.333% task-bootstrap intervals.

Contrast	Effect (pp)	95% CI	Simultaneous CI	Discordant L/R	Holm p
B minus C: address	0.0	$[0, 0]$	$[0, 0]$	0/0	1.000
A minus B: local reassertion	-15.0	$[-27.5, -5.0]$	$[-30.0, -2.5]$	0/6	.09375
B minus D: subtree/root	$+12.5$	$[2.5, 22.5]$	$[2.5, 25.0]$	5/0	.125

The apparent tension between bootstrap intervals and adjusted tests is expected with only five or six discordant pairs: percentile intervals and exact sign tests are different procedures, and familywise correction is decisive under the frozen rule. We report both rather than selecting the more favorable summary.

Failure taxonomy over all 160 rows is 149 exact, one invalid envelope, eight invalid nested replacements, one invariant rejection, and one timeout. All six A failures are invalid nested scalar JSON strings. D contributes two invalid nested replacements, one invalid envelope, one invariant rejection, and the timeout. Median completion tokens are 88.5, 117.5, 116.5, and 453 for A–D; reported reasoning tokens are zero. Median latencies are 5.02, 5.35, 5.40, and 9.52 seconds.

G.4 REPRESENTATION-BY-SCOPE DIAGNOSTIC FOLLOW-UP

The follow-up was designed after observing the original six A failures and is not an independent population replication. It reuses the 40 source documents but deterministically selects fresh edit intents: 39 target a different path, and the only document with one eligible leaf receives a different replacement value. The factors are resolved leaf versus resolved parent scope and JSON-in-string versus native-JSON replacement representation. The full root, correct intent, outer keys, hash copying and runtime verification, structural invariant, model settings, worker blocking, and Latin-square order remain fixed.

The single primary contrast is leaf-native minus leaf-string. It receives a paired 95% bootstrap interval and a two-sided exact sign/McNemar test without multiplicity adjustment. Parent representation, scope under native representation, and scope under string representation form a Holm-adjusted secondary family; the difference-in-differences is descriptive. Before calls we recorded that six perfectly one-directional primary discordances give exact $p = .03125$, five give $p = .0625$, and the smallest secondary Holm test requires seven. Joint ceiling or floor is saturation-limited, not equivalence.

The 160 treatment assignments were attempted exactly once, with no retry or reissue. All endpoint values were recomputed from the returned candidates.

Table 11: Unconditional fresh-intent factorial endpoints and operational medians. LS/LN are leaf-string/native; PS/PN are resolved-parent string/native.

Arm	Exact	Contract	Candidate	Completion tokens	Latency (s)
LS: leaf, JSON-in-string	39/40	39/40	39/40	88.0	5.19
LN: leaf, native JSON	40/40	40/40	40/40	97.0	5.30
PS: parent, JSON-in-string	40/40	40/40	40/40	113.5	5.46
PN: parent, native JSON	39/40	39/40	40/40	138.0	5.78

The primary leaf-native minus leaf-string contrast is $+2.5$ points (95% CI $[0, 7.5]$; one positive discordance; exact $p = 1$). No secondary contrast is detected after Holm adjustment. The descriptive interaction is $+5.0$ points (CI $[0, 15.0]$).

The explicitly post-hoc failure audit finds that both failures share one task. Its JSON key and old scalar value are both `./*`, and the target value is `./*-ownership`. LS returns that text without inner JSON quotes, while PN renames the key instead of changing its value; the topology invariant rejects the latter. The other 39 tasks succeed in all four arms. This concentration does not support a stable representation or scope effect and shows why the original A–B contrast should not be read as carry-forward in isolation.

G.5 Matched-scope volume stress test

The volume study was motivated by a two-document exploratory pilot and is not an independent population replication of the public-source corpus. Its formal design compares two providers that could satisfy the common response envelope. Pilot outcomes do not enter inferential tests, and no conclusion is drawn about other providers.

A deterministic random base64url carry-forward field is inserted inside the target parent; the 512-character value is an exact prefix of the 24,576-character value. For each provider and volume, leaf and parent arms receive the identical authoritative root once, the correct absolute edit intent, and a runtime-resolved scope. Leaf returns the successor scalar, after which the runtime carries forward the root. Parent returns the complete successor parent, including the unchanged probe, and the runtime splices it into the same root. Both use a native-JSON three-key envelope, copied scope hash, and the same fail-closed evaluator. Thus the within-volume contrast fixes input, address, and commit while assigning the unchanged probe to runtime inheritance in the leaf arm and to generative exposure Γ_d in the parent arm.

All 320 scheduled provider–task–scope–volume attempts enter the formal analysis, with one attempt per prompt and no retry. Four MiniMax attempts timed out without a returned candidate; the remaining 316 returned responses under the requested model family.

Table 12: Matched-scope volume endpoints. Payload is evaluator-confirmed equality of the injected carry-forward value after candidate construction; unavailable or unparsed candidates count false. Payload and Contract are separate endpoints; neither nests the other. Tokens and latency are medians over all rows in the arm.

Provider	Scope	Payload size	Exact	Contract	Payload	Completion tokens	Latency (s)
MiniMax-M3	leaf	512	38/40	38/40	38/40	64.5	4.91
MiniMax-M3	leaf	24,576	40/40	40/40	40/40	65.0	5.91
MiniMax-M3	parent	512	36/40	36/40	39/40	432	8.79
MiniMax-M3	parent	24,576	19/40	25/40	19/40	16,285.5	119.21
Qwen3.8-27B	leaf	512	40/40	40/40	40/40	98.0	4.28
Qwen3.8-27B	leaf	24,576	40/40	40/40	40/40	97.0	6.62
Qwen3.8-27B	parent	512	39/40	39/40	39/40	521	9.09
Qwen3.8-27B	parent	24,576	29/40	35/40	30/40	18,096	210.10

For task i, provider p, and volume v, let $g_{ip}(v) = Y_i(\text{leaf}, v) - Y_i(\text{parent}, v)$ and $h_{ip} = g_{ip}(24\text{K}) - g_{ip}(512)$. Table 13 follows the frozen serial hierarchy. Qwen's provider-specific interaction is descriptive, although its direction and analysis were fixed in advance.

Table 13: Volume-stress contrasts. P/N/T are positive/negative/tied task contrasts.

Contrast	Effect (pp)	95% CI	P/N/T	Exact p	Role/status
G: MiniMax parent, 512 minus 24K	$+42.5$	$[25.0, 60.0]$	18/1/21	.0000381	gate, passed
P1: MiniMax h	$+47.5$	$[32.5, 62.5]$	19/0/21	.00000381	primary, passed
Qwen h	$+25.0$	$[12.5, 40.0]$	10/0/30	.001953	descriptive
P2: MiniMax h minus Qwen h	$+22.5$	$[2.5, 42.5]$	13/4/23	.04904	heterogeneity, boundary pass

G and P1 are members of the same MiniMax contrast family, not independent replications. At the aggregate-rate level,

$$\text{P1} = \{L_{24K} - P_{24K}\} - \{L_{512} - P_{512}\} = \text{G} + \{L_{24K} - L_{512}\} = 42.5 + 5.0 = 47.5 \text{ points.}$$

Their exact p-values differ because G uses the prespecified one-sided sign test on parent low-minus-high contrasts (18/1), whereas P1 uses a two-sided test on task-level interactions (19/0). Serial gatekeeping supplies an error-control order rather than a second source of evidence.

MiniMax's 21 high-parent exact failures comprise ten schema rejections, six accepted-but-wrong returns, four timeouts, and one invalid envelope. Qwen's 11 comprise six accepted-but-wrong returns, three schema rejections, and two invalid envelopes. Every high-volume leaf return is exact. A post-run sensitivity audit gives a P2 leave-one-task-out p range of .0213–.0768: deleting any one of its 13 positive discordances makes $p > .05$, whereas P1 remains below .000008 under every one-task deletion. Removing the earlier pathological `./*` task changes neither discordant count. Accordingly P1 is the strong matched-scope result; P2 is fragile evidence about magnitude heterogeneity, not a universal provider ordering.

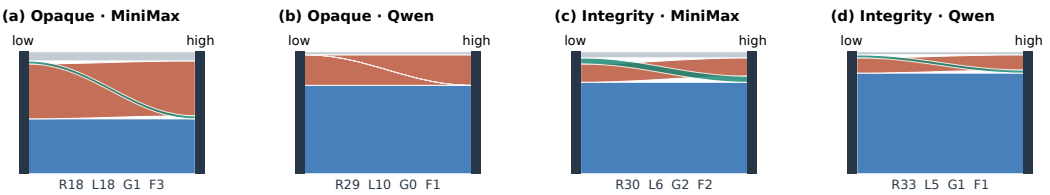


Figure 5: Paired low-to-high parent-scope exactness transitions for all four substrate–provider blocks ($n = 40$ tasks each). Blue stays exact, rust loses exactness, teal gains exactness, and gray fails at both doses. R/L/G/F below each panel count retained exact, lost, gained, and failed-both tasks. The opaque MiniMax flow is the prespecified stress-test block; other flows describe separate configurations rather than replications of its effect.

The complete 24,576-character payload appears exactly once in every high-volume input, so the leaf arm reads the same large authoritative state rather than a short substitute. No completed high-parent response reports output-cap termination. Excluding the four MiniMax timeout blocks leaves P1 at $+44.4$ points (CI $[27.8, 61.1]$, $p = .0000305$); counterfactually scoring all four timeouts exact leaves $+37.5$ points (CI $[20.0, 55.0]$, $p = .000275$). These are sensitivity analyses on the same paired tasks.

Table 14 audits the retained candidate strings without adding an endpoint or provider call. Every contract-accepted but inexact parent-24K candidate corrupts the payload. Most errors are single-character changes rather than truncation; the first mismatch varies widely across the string. The only Qwen exact failure with the payload intact in both snapshots has a correct target leaf but changes three sibling array paths and is rejected for a topology violation. It is neither a leaf-edit error nor payload corruption.

Table 14: Post-run forensics for contract-accepted but inexact parent-24K candidates. "One char" combines one-character insertion, deletion, and substitution; "longer" denotes a multi-character local change. No row loses the full payload or ends by prefix/suffix truncation.

Snapshot	Provider	Wrong	Payload corrupt	One char	Longer	First mismatch range
Original	MiniMax	6	6	5	1	3,921–21,402
Original	Qwen	6	6	4	2	2,311–21,922
Relaxed	MiniMax	11	11	10	1	6,248–21,995
Relaxed	Qwen	7	7	6	1	1,183–22,839

G.6 Structured-list transfer with lockfile fields

This follow-up was designed after the random-string forensics and fixed before any provider call. It replaces one generated base64url scalar with 486 verbatim 95-character SRI integrity values from a public npm lockfile. A task-specific permutation and nested prefixes form native-JSON lists of 491, 8,135, 16,367, and 24,501 canonical characters. No value is generated, truncated, padded, or selected using an outcome. SRI strings are themselves opaque base64-encoded hashes; relative to the earlier study, the salient design change is also organization from one long scalar to many 95-character items. The list remains experimentally aggregated and inserted into another public JSON parent, not an untouched production lockfile.

The same 40 document–intent blocks are reused to hold task heterogeneity fixed. Each block crosses two providers, two scopes, and four doses, for 640 one-shot attempts in a cyclic 16-treatment order. No formal timeout or transport failure occurs. This later provider snapshot is neither a new task population nor a causal contrast of list versus scalar content with the earlier study.

Table 15: Aggregated lockfile-SRI exact commits over 40 tasks. Column labels are realized canonical payload characters; nominal targets were 512/8K/16K/24K.

Provider	Scope	491	8,135	16,367	24,501
MiniMax-M3	leaf	40	38	39	39
MiniMax-M3	parent	36	32	29	32
Qwen3.8-27B	leaf	40	40	40	40
Qwen3.8-27B	parent	38	35	35	34

The frozen gate, MiniMax parent 512 minus 24K, is $+10.0$ points with CI $[-2.5, 22.5]$: six task blocks favor low, two favor high, and 32 tie (one-sided exact $p = .1445$). The primary task-level slope of $g_i(L) = Y_i(\text{leaf}, L) - Y_i(\text{parent}, L)$ over $\log L$ is positive descriptively but has 12 positive, five negative, and 23 tied slopes ($p = .1435$; bootstrap CI crosses zero). Qwen's ordered replication has five positive, one negative, and 34 tied slopes ($p = .2188$). Because the gate fails, neither downstream test is a hierarchical pass. The descriptive 512-to-24K scope interactions are $+7.5$ points for MiniMax (CI $[-7.5, 22.5]$, $p = .508$) and $+10.0$ for Qwen ($[0, 22.5]$, $p = .219$).

The run-frozen plan specified tests and directions but no prospective MDE or tie-rate model. Under the one-sided gate, four ideal all-positive discordances give $p = .0625$ and five give $p = .03125$; at 40 pairs the smallest such passing margin is therefore 12.5 points. The observed effective sample sizes are only 8, 17, and 6 discordant tasks for G/P1/P2. Hierarchy failure means not detected under this design, not equivalence or a powered rejection.

MiniMax's four leaf misses are all pre-commit contract failures with no candidate: one 8K response drops two hash characters, while one Node-RED task returns numeric `2` instead of string `"2"` at 8K, 16K, and 24K. Qwen leaf is 160/160. Thus leaf allocation is not unconditionally reliable, but none of these misses is carry-forward corruption.

Table 16: Directly re-extracted SRI-payload preservation among candidate-available parent returns. Payload and contract are separate endpoints; neither nests the other.

	491		8,135		16,367		24,501	
Provider	Cand.	Pres.	Cand.	Pres.	Cand.	Pres.	Cand.	Pres.
MiniMax-M3	38	37	37	34	37	30	39	35
Qwen3.8-27B	40	39	40	37	40	37	40	36

A post-hoc reinspection of all 311 candidate-available parent successors found 26 list-payload errors: 25 omit the list or replace it with a non-list value, and one rewrites a whole integrity item. None reorders items or introduces a one-character list-item error. The corrected secondary classification does not change exact outcomes or the prespecified contrasts. These list failures differ qualitatively from the mostly single-character errors in the opaque-string study, so a shared independent-character error model is not supported.

Median parent completion tokens rise from 403 to 15,502 for MiniMax and from 515 to 18,381 for Qwen; corresponding median latencies rise from 5.32 to 63.15 seconds and 8.11 to 211.54 seconds. Leaf medians stay at 65–99 tokens and 3.3–4.4 seconds. Runtime inheritance removes these unchanged fields from required model output, but exact reliability does not trace the prespecified portable curve. Completion tokens are an observed cost endpoint, distinct from the source-region measure $|\Gamma_d|$.

G.7 Relaxed-budget parent-24K diagnostic

This post-hoc diagnostic reuses the identical 40 parent-24K tasks and evaluators, increasing the requested output budget from 32,768 to 65,536 tokens and the outer deadline from 300 to 900 seconds. Its 80 one-shot attempts span a later provider snapshot; the comparison therefore tests robustness of the observed deficit under a looser configuration without identifying a pure budget effect.

Table 17 reports unconditional endpoints for all 80 attempts. Thirty-one of 34 failures occur after a provider response returns. No completed response reports termination at the output cap; maximum observed completion usage is 19,426 tokens for Qwen and 17,347 for MiniMax.

Table 17: Relaxed-budget parent-24K endpoints. Payload uses the evaluator-confirmed definition above. Failure counts remain in the unconditional denominator.

Provider	Exact	Contract	Payload	Timeout	Transport	Median tokens	Returned failure classes
Qwen3.8-27B	28/40	35/40	29/40	0	0	18,089	7 wrong; 5 schema
MiniMax-M3	18/40	29/40	18/40	2	1	16,329	11 wrong; 7 schema; 1 envelope

The two MiniMax timeouts end near the 900-second deadline; the transport failure returns no candidate. None is a near-complete returned response.

Qwen changes from 29/40 in the original parent-24K arm to 28/40 relaxed: one task improves, two deteriorate, and 37 tie (paired change -2.5 points, CI $[-12.5, 5.0]$, exact $p = 1$). MiniMax changes from 19/40 to 18/40: seven improve, eight deteriorate, and 25 tie (-2.5 points, $[-22.5, 17.5]$, $p = 1$). Its transition matrix is bidirectional rather than monotone recovery. Relative to the original 40/40 leaf-24K arms, residual historical gaps are 30.0 points for Qwen and 55.0 for MiniMax; these are descriptive cross-snapshot point estimates, so we report no interval or p-value. Even treating MiniMax's three calls without responses as exact leaves 21/40.

Because these task-paired comparisons span a later provider snapshot, they do not causally identify deadline or token-cap effects. Their appropriate conclusion is narrower: the parent-24K deficit did not disappear under the run-frozen looser-budget configuration, and observed cap termination is not the dominant failure type. Untouched production state and matched compute remain open replications.

H CANDIDATE FIDELITY BEYOND RELEASED TESTS

All 320 task–arm trials are analyzed. The 182 available candidates are each evaluated with the corresponding official QuixBugs test module. Candidate-unavailable trials remain failures. Table 19 gives unconditional paired intervals and discordances.

Table 18: Historical QuixBugs outcomes per 40 attempts. P/R: custom runtime Patch / full-file Rewrite. Official tests and exact file equality are separate endpoints. These model configurations were evaluated before the new same-task method study.

Configuration	Contract P/R	Exact P/R	Tests P/R	Exact gap	Test gap
DeepSeek-V4-Flash	0/34	0/17	0/34	-42.5	-85.0
MiniMax-M3	13/8	13/4	13/8	$+22.5$	$+12.5$
Qwen2.5-Coder-1.5B Q4	12/40	12/10	12/10	$+5.0$	$+5.0$
Qwen3.8-27B	40/35	40/21	40/35	$+47.5$	$+12.5$

Table 19: Unconditional Patch-minus-Rewrite gaps. Discordance is Patch-only / Rewrite-only success.

Configuration	Exact gap [95% CI]	Exact discord.	Semantic gap [95% CI]	Semantic discord.
DeepSeek-V4-Flash	$-42.5[-57.5, -27.5]$	0/17	$-85.0[-95.0, -72.5]$	0/34
MiniMax-M3	$+22.5[7.5, 37.5]$	11/2	$+12.5[-7.5, 32.5]$	11/6
Qwen2.5-Coder-1.5B Q4	$+5.0[-15.0, 25.0]$	10/8	$+5.0[-15.0, 25.0]$	10/8
Qwen3.8-27B	$+47.5[32.5, 62.5]$	19/0	$+12.5[2.5, 22.5]$	5/0

Table 20: Post-treatment conditioned diagnostics. "None accepted" replaces the ambiguous historical notation 0/0. P/R success counts use the selected denominator.

Configuration	Patch-accepted (n)	Exact P/R	Semantic P/R	Both-accepted (n)	Semantic P/R
DeepSeek-V4-Flash	0 (none accepted)	–	–	0 (none accepted)	–
MiniMax-M3	13	13/2	13/2	2	2/2
Qwen2.5-Coder-1.5B Q4	12	12/2	12/2	12	12/2
Qwen3.8-27B	40	40/21	40/35	35	35/35

For DeepSeek Rewrite, all 34 available candidates pass official tests and are AST-equivalent to target although only 17 are byte-exact. MiniMax has 13 available Patch and eight available Rewrite candidates; all 21 pass. Local Qwen2.5 has 12 available Patch and all 40 Rewrite candidates, but only 12 and 10 pass respectively. Qwen3.8 has 40 Patch and 35 Rewrite candidates; all pass and are AST-equivalent. These candidate-conditional facts diagnose fidelity and acceptance but do not erase system-level contract failure.

I SAME-TASK PUBLISHED EDIT-INTERFACE ADAPTATIONS

The same 40 QuixBugs Python tasks, authoritative source files, and correct character-span intents are shared across delivery interfaces. The correct replacement text and its ordered character addresses are supplied. The deterministic oracle verifies the source hash and old values, applies replacements right to left, and uses no model call. It produces all 40 canonical files on QuixBugs and the selected HumanEvalFix-Python set. This is an upper bound for the conditional delivery setup, not an autonomous repair method. The five-format core compares runtime-owned Patch, full-file Rewrite, Agentless repair-stage SEARCH/REPLACE, and untuned BlockDiff and FuncDiff prompts derived from AdaEdit. A fixed task order balances within-task arm position. Each task–arm pair receives one greedy call with a 4,096-token output cap; failed formats and unavailable candidates remain in the denominator. The 1.5B, 7B, and 14B checkpoints use the same 4-bit local inference configuration and are analyzed separately.

These rows isolate delivery interfaces, not complete published agents. Agentless search and candidate selection are supplied by the common task design; AdaEdit's trained format specialists and selector

are not evaluated. The CODESTRUCT extension tests its documented exact-string small-edit tool, with a unique-match requirement, rather than its full AST agent. The SWE-agent extension tests one ACI line-range action with an editor/lint gate, rather than interactive exploration and retry. Candidate availability, official-test pass, and byte-exact target equality are unconditional and distinct outcomes. Every formal task receives one attempt, and the same official task tests are applied to accepted files.

The added Unified Diff path uses strict `git apply`; SWE-bench motivates the format but is a benchmark, not an implemented repair agent. Agentless uses its repair-stage `SEARCH/REPLACE` parser. AdaEdit-B/F use the BlockDiff/FuncDiff specifications and official appliers with untuned format prompts. The CODESTRUCT and SWE-agent paths are released auxiliary tool actions rather than their full agent trajectories. The 32B comparison caps generation at 4,096 tokens and 180 seconds; its output-token and GPU-time medians include rejected calls, while time excludes parsing, tool execution, and tests.

Table 21: Frozen five-format local outcomes, each over the same 40 tasks. Candidate means that the requested interface produced an applicable file; Tests denotes the official task test module, and Exact denotes byte equality to the frozen target.

Checkpoint	Interface	Candidate	Tests	Exact
1.5B	Full Rewrite	7/40	3/40	1/40
1.5B	BlockDiff prompt (untuned)	0/40	0/40	0/40
1.5B	FuncDiff prompt (untuned)	0/40	0/40	0/40
1.5B	Agentless SEARCH/REPLACE	2/40	0/40	0/40
1.5B	Runtime Patch	34/40	32/40	31/40
7B	Full Rewrite	40/40	37/40	19/40
7B	BlockDiff prompt (untuned)	0/40	0/40	0/40
7B	FuncDiff prompt (untuned)	0/40	0/40	0/40
7B	Agentless SEARCH/REPLACE	30/40	29/40	29/40
7B	Runtime Patch	36/40	36/40	36/40
14B	Full Rewrite	40/40	38/40	29/40
14B	BlockDiff prompt (untuned)	0/40	0/40	0/40
14B	FuncDiff prompt (untuned)	0/40	0/40	0/40
14B	Agentless SEARCH/REPLACE	38/40	38/40	38/40
14B	Runtime Patch	40/40	40/40	40/40

Tool and format failure modes. The CODESTRUCT exact-string tool yields 27/40, 34/40, and 38/40 applicable candidates at 1.5B, 7B, and 14B. At 1.5B, three responses fail the action contract, nine name absent old text, and one uses a nonunique old string; the 7B failures are four nonunique and two absent anchors, and the 14B failures are two nonunique anchors. These are failures of action selection or applicability, not evidence about the full AST system.

The untuned BlockDiff and FuncDiff rows fail their strict native format contract under prompt-only weights. Their zero counts do not estimate AdaEdit's separately trained format specialists. An exploratory permissive-diff interpretation parses more responses but produces no byte-exact files; changing the acceptance contract after observing outputs does not revise the primary score.

SWE-agent ACI action path. At 7B, all 40 one-action outputs meet the action grammar. The editor/lint gate rejects five: three syntax or indentation errors and two undefined-name errors. Of the 35 applicable candidates, 33 pass official tests and 30 equal the canonical target. Three test-passing noncanonical files each delete one blank line adjacent to the edit range. These outcomes concern a single supplied-intent action, not a full interactive agent.

Why exact match and finite tests differ. For the 7B full-file Rewrite arm, 19/40 files are byte exact, whereas 37/40 pass official tests. Seventeen of the 18 test-passing noncanonical files delete only a leading blank line from the target. Ignoring that line *post hoc* would raise the exact count to 36/40, without changing the predeclared endpoint. In the remaining case, the rewrite changes the order of empty-input and zero-total guards. It passes the finite benchmark tests but returns 0 for an

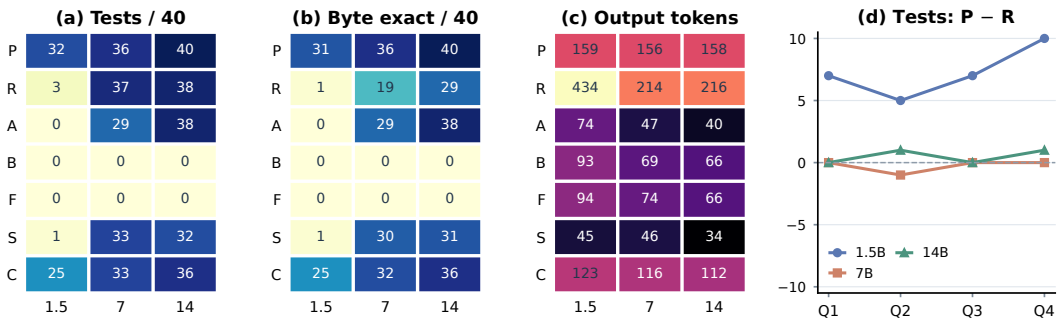


Figure 6: Verified method diagnostics across 1.5B, 7B, and 14B on the same 40 tasks. (a,b) Official-test and byte-exact successes; (c) median completion tokens over all 40 attempts, including rejected outputs; (d) within-quartile Patch-minus-Rewrite test-pass counts, with ten files per source-length quartile. P/R/A/B/F/S/C denote runtime Patch, full Rewrite, Agentless, untuned BlockDiff, untuned FuncDiff, SWE-agent ACI edit, and the auxiliary CODESTRUCT string tool. Panel (d) is post-hoc descriptive; the tool-action arms were scheduled separately.

empty coin list and zero total, where the canonical target returns 1. This is a concrete counterexample to treating finite-test pass as semantic equivalence.

Post-hoc source-length strata. For a descriptive size check, the 40 frozen files are sorted by authoritative source character count, breaking ties by task ID, and split into four groups of ten: Q1 spans 291–477 characters, Q2 509–573, Q3 610–881, and Q4 923–1,711. We count official-test successes within each group for Patch, Rewrite, and Agentless under each separately pinned checkpoint. This grouping was chosen after the primary studies and is not a prespecified effect-modification test.

Table 22: Paired Runtime Patch minus Full Rewrite on the same 40 tasks at each checkpoint. Differences and task-bootstrap 95% intervals are in percentage points; $n_+ : n_-$ counts discordant tasks favoring Patch versus Rewrite. These are model-specific descriptive intervals, without pooling across scales.

Checkpoint	Tests Δ [95% CI]	Tests $n_+ : n_-$	Exact Δ [95% CI]	Exact $n_+ : n_-$
1.5B	$+72.5\ [55.0, 87.5]$	$31 : 2$	$+75.0\ [60.0, 90.0]$	$31 : 1$
7B	$-2.5\ [-12.5, 7.5]$	$2 : 3$	$+42.5\ [25.0, 60.0]$	$19 : 2$
14B	$+5.0\ [0.0, 12.5]$	$2 : 0$	$+27.5\ [15.0, 42.5]$	$11 : 0$

The quartile grouping and task-bootstrap intervals are descriptive and were not used to select the primary comparison.

Table 23: Exploratory 7B paired Runtime Patch minus other method paths on the same 40 tasks. Differences and task-bootstrap 95% intervals are percentage points; $n_+ : n_-$ counts discordant tasks favoring Patch versus the comparator. Intervals are descriptive and unadjusted for multiple contrasts; the two tool-action paths were scheduled separately. The Rewrite contrast appears in Table 22.

Comparator	Tests Δ [95% CI]	Tests $n_+ : n_-$	Exact Δ [95% CI]	Exact $n_+ : n_-$
Agentless SEARCH/REPLACE	$+17.5\ [2.5, 32.5]$	$9 : 2$	$+17.5\ [2.5, 32.5]$	$9 : 2$
SWE-agent ACI edit	$+7.5\ [-5.0, 20.0]$	$5 : 2$	$+15.0\ [2.5, 27.5]$	$7 : 1$
CODESTRUCT aux. str_replace	$+7.5\ [-2.5, 20.0]$	$4 : 1$	$+10.0\ [0.0, 22.5]$	$5 : 1$

These task-level comparisons are descriptive and do not identify an end-to-end agent effect. The tool-action extensions were run in separate sessions, which should be considered when reading small differences.

I.1 MATCHED 32B BF16 QUIXBUGS EXTENSION

The five-format core and two separately scheduled tool-action extensions use the same pinned Qwen2.5-Coder-32B-Instruct weights (Hui et al., 2024), BF16 precision on an H20 GPU, greedy decoding, a 4,096-token cap, and a 180-second model deadline. All 280 one-shot attempts finish without inference or scoring error. Independent replay checks every raw response through the corresponding parser or tool and reruns the official tests on applicable candidates. The two action paths preserve the frozen task set and gold edit intent; their separate scheduling is relevant when reading small differences. The model size, precision, backend, and hardware differ from the earlier 4-bit scale study.

On QuixBugs, SWE-agent's single ACI `edit` yields 39/40 applicable candidates, 38/40 test passes, and 37/40 byte-exact files. Its remaining response is rejected by the released editor. CODESTRUCT's auxiliary `str_replace` yields 38/40 candidates, 36/40 test passes, and 36/40 byte-exact files; one old string is nonunique and another is absent. These are one-action delivery paths, not full interactive agent estimates. The main table retains the relevant tool actions; untuned AdaEdit grammar prompts remain in appendix diagnostics.

Patch passes all 40 tests and yields all 40 canonical files; Rewrite reaches 39 test passes but only 24 exact files. Paired Patch-minus-Rewrite differences are $+2.5$ points on released tests (descriptive 95% task-bootstrap interval $[0, 7.5]$, discordance $1 : 0$) and $+40.0$ points on exact files ($[25, 55]$, discordance $16 : 0$). A post-hoc audit attributes Rewrite's 15 test-passing noncanonical files to 14 deleted leading blank lines and one terminal-line-ending difference. The frozen byte-exact endpoint remains unchanged. The untuned BlockDiff and FuncDiff prompts give no applicable candidates under their strict appliers; this does not estimate AdaEdit's trained editors.

Figure 7 preserves the paired task structure behind the aggregate counts. At 7B, test-passing noncanonical Rewrite files concentrate in tasks where Patch is byte exact, whereas the two Rewrite-only exact tasks also appear as local Patch failures. The display is descriptive; it does not turn those task groups into independent subpopulations.

J HUMANEVALFIX-PYTHON FIXED-INTENT SETUP

We derive a delivery task from the Python repair examples released with HumanEvalPack (Muennighoff et al., 2024). The authoritative file is the exercise prompt followed by its buggy solution; the canonical target uses the same prompt followed by the released correction. The complete target is withheld from the model, while the correct old and new edit spans are supplied to every interface. This transforms the original autonomous repair task into a conditional test of edit delivery.

Of the 164 released Python examples, 22 lack a unique contiguous line replacement under these criteria. The remaining 142 buggy files fail and their canonical targets pass the released tests; nine fail the requirement that gold actions be applicable through every one of the seven interfaces. We order the 133 eligible task IDs by a fixed hash and select the first 40 before model inference. They contain 38 one-line and two two-line replacements; authoritative files span 140–1,173 characters. This selection makes the comparison paired and interface-applicable, while limiting its population claim to short localized repairs.

All seven paths receive the same file and correct edit intent: runtime-owned local Patch, full-file Rewrite, Agentless SEARCH/REPLACE, the untuned BlockDiff and FuncDiff format prompts, one SWE-agent ACI edit, and the CODESTRUCT toolkit's auxiliary exact-string replacement. Under the pinned 32B BF16 checkpoint, each task–path pair receives one greedy call, a 4,096-token output cap, a 180-second model deadline, and no retry. A task rotation gives every path five or six appearances at each of seven within-task positions. A candidate is scored against the released test containing `check(entry_point)` and against the canonical file bytes. Invalid or unavailable candidates remain failures in all unconditional counts. These paths do not reproduce localization, trained format selection, or full interactive agent trajectories.

Figure 8 shows the task-level and cost diagnostics for these paths.

On this selected set, local Patch yields 40/40 applicable, test-passing, and byte-exact successors. Full Rewrite yields 40/40 candidates, 39/40 test passes, and 26/40 byte-exact successors; Agentless

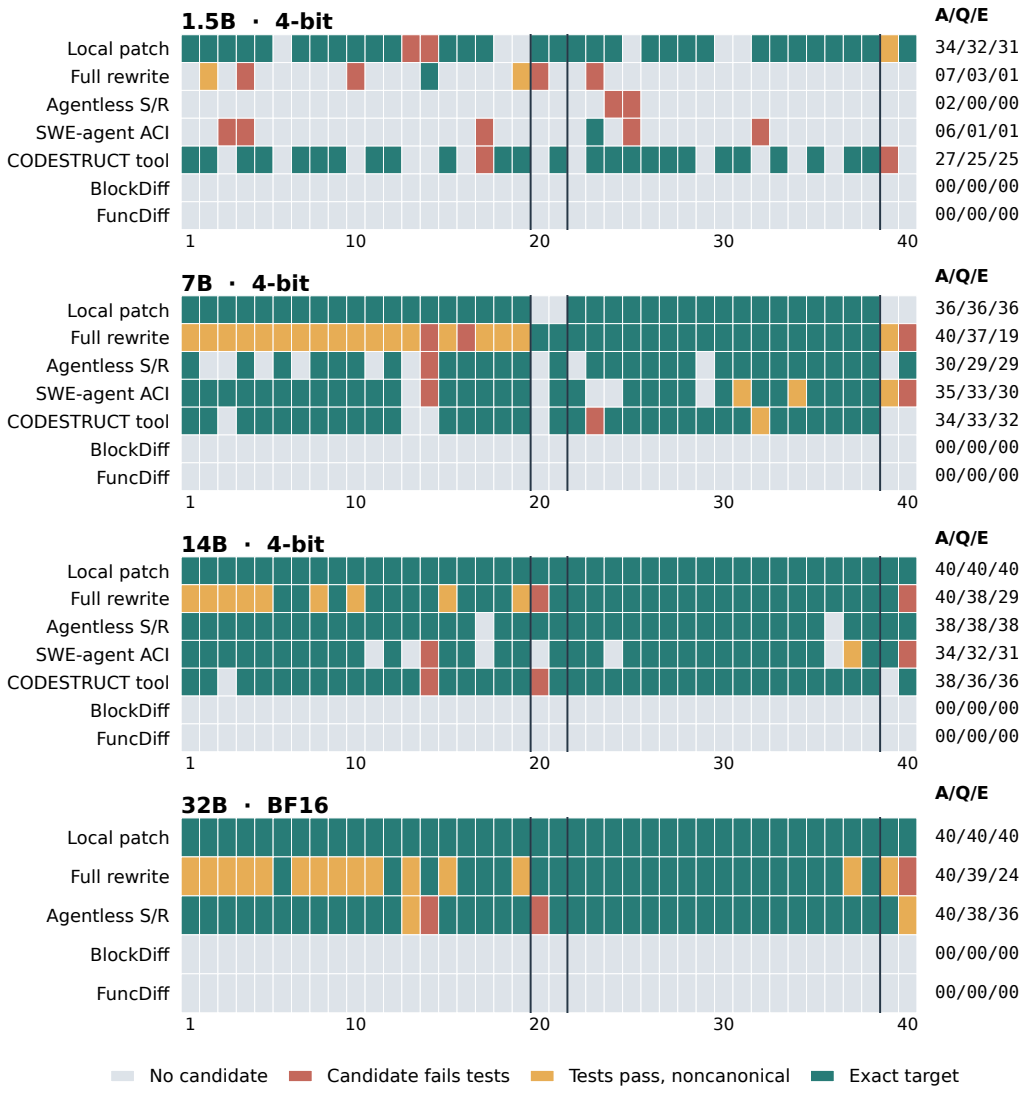


Figure 7: Task-level outcomes for four separately analyzed local deployments. Each cell is one of the same 40 QuixBugs tasks. Columns are ordered by the 7B Patch/Rewrite exact-success pattern: Patch only (19), Rewrite only (2), both (17), neither (2), with alphabetical order within group; dark lines mark group boundaries. Colors partition unavailable candidate, candidate that fails tests, test-passing noncanonical file, and byte-exact target. Right-side A/Q/E are candidate availability, official-test pass, and byte-exact candidate counts, matching the main-text notation. The 32B panel uses different precision and hardware; the untuned BlockDiff and FuncDiff rows are prompt-only adaptations.

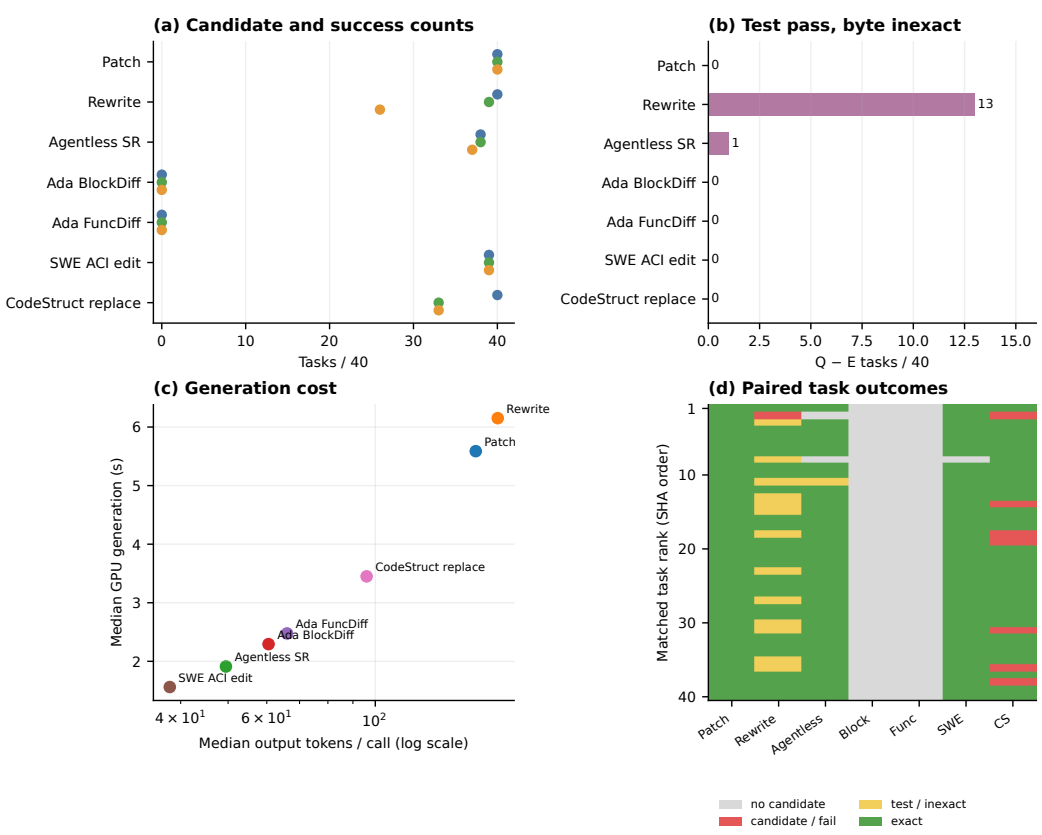


Figure 8: HumanEvalFix-Python diagnostics from the independently verified 32B BF16 ledger, with 40 paired tasks per method. (a) Applicable candidates A (blue), released-test passes Q (green), and byte-exact files E (orange); (b) test-passing but byte-inexact counts $Q - E$; (c) median completion tokens versus median GPU generation latency, including rejected responses; (d) task-level outcomes under the seven delivery paths, ordered by a fixed task-ID hash. Gray is no candidate, red is candidate failing tests, yellow is test-passing noncanonical, and green is exact. The format-prompt rows use untuned weights; all panels are exploratory diagnostics rather than a pooled system-level comparison.

yields 38/40, 38/40, and 37/40; the SWE-agent one-action path yields 39/40 on all three; and the CODESTRUCT auxiliary string path yields 40/40, 33/40, and 33/40. Neither untuned AdaEdit format prompt produces an applicable candidate under its strict applier. These are outcomes of one-call delivery paths on oracle-intent tasks, not estimates of the published systems' original autonomous repair rates.

The 13 test-passing but byte-inexact Rewrite candidates are a useful endpoint sensitivity check. Twelve remove exactly one leading blank line from the canonical file; the remaining one changes a quotation mark inside a docstring example. Under Python AST equality, 38/40 rewrites match their canonical targets, versus 40/40 for local Patch. AST equality is a static structural diagnostic, not a proof of behavioral equivalence. The released-test gap is only one task, so the larger exact-file gap should be interpreted mainly as a difference in preservation of lexical state.

J.1 All-eligible extension and endpoint audit

To test sensitivity to the 40-task hash selection, we froze and ran every task satisfying the original common-interface eligibility criteria. The complete funnel is shown below; this still excludes tasks that require noncontiguous or interface-inapplicable gold edits. The first 40 task records and prompts coincide with the original subset, but the 133-task results arise from a separate inference schedule, not additional independent samples for those first 40 tasks.

Table 24: HumanEvalFix-Python selection funnel. Criteria were applied before either model run.

Stage	Remaining	Exclusion at stage
Released Python examples	164	—
Unique contiguous-line replacement	142	22 lack one representable span
Common gold action and test gate	133	9 fail an interface applicability gate
Original hash-selected subset	40	93 eligible tasks held out from the initial study

The extension contains 133 tasks per each of seven original delivery paths, totaling 931 model calls; a separately scheduled strict Unified Diff path adds 133. Independent replay verifies all raw responses and reruns released tests for all 660 applicable original-path and 128 applicable diff candidates. Direct deterministic application also produces all 133 targets with zero model calls. Table 2 reports the paths with meaningful applicable candidates; the two untuned AdaEdit grammar prompts again produce none and remain format diagnostics rather than trained-method comparisons. Among Rewrite's 51 test-passing, byte-inexact files, 47 remove the leading newline, one changes only the terminal newline, one differs lexically while retaining the target AST, and two have a different AST despite passing the released tests. Thus 130/133 Rewrite files match the target AST, versus 81/133 matching target bytes. The 133-task comparison strengthens the selected-set precision for lexical fidelity but does not enlarge the task family beyond localized, oracle-intent Python edits.

Runtime-owned descriptor ablation. An additional frozen 133-call run uses the same task pack, pinned weights, and one-shot decoding, but the runtime retains the correct checked descriptor D^* while the model returns only a JSON array of replacement values. Source-hash and old-value checks still precede splicing. Independent replay gives $A/Q/E/\mathrm{AST} = 132/132/132/132$; one fenced response contains invalid JSON and is rejected. Median completion falls from 160 to 28 tokens and model generation from 5.43 to 1.17 seconds relative to the original Patch envelope; directly logged model-through-test wall time is 1.23 seconds per median trial. This is a separate schedule, so the cost ratios are descriptive. Against SWE-agent's single edit action, the paired byte-exact discordance is $2:1$ (two-sided exact sign $p = 1$), while the value-only output is shorter. Against the original Patch envelope it loses one task. Direct application of the given values remains the zero-model upper bound. This ablation changes the return encoding e_d, not B_d or Γ_d, and shows that redundant descriptor echo was a major output-cost choice rather than a necessary part of source-grounded composition.

Table 25 reports the task-level spread behind the Tok and Gen medians in the main comparison. The quartiles describe heterogeneity across the 40 fixed tasks; they do not measure variation from repeated model sampling or independent training runs.

J.2 Strict unified-diff grammar audit

SWE-bench's direct patch-generation setting motivates the strict unified-diff control (Jimenez et al., 2024); this is a grammar transfer, not SWE-bench's issue-to-repair task. A generated zero-context diff must apply without correction to the same frozen source and correct intent as Table 1. It has nine QuixBugs and two HumanEvalFix patch-application rejections. Against local Patch, its paired exact-file discordances are $0:9$ and $0:3$, respectively (two-sided exact McNemar $p = .0039$ and $.25$). Compact output can therefore fail at strict application even when the intended edit is supplied.

K Source-Frame and Structural Audit

We re-examine the same 32B candidates with two post-hoc endpoints. Frame preservation U asks whether a candidate can be represented by replacing only the frozen writable span or spans while retaining every complementary source segment verbatim. All selected HumanEvalFix files and 39 QuixBugs files have one writable span; the remaining QuixBugs file has two. AST agreement asks whether Python parsing yields the same abstract syntax tree as the canonical target, disregarding formatting but not literal values. Both endpoints use all 40 attempts per path, counting missing candidates as failures. AST agreement is a structural diagnostic, not proof of program equivalence.

Table 25: Across-task dispersion for the fixed-intent comparison (40 matched tasks per dataset and path). Cells report median with $[Q_1, Q_3]$ across tasks, not variation across random seeds. Tok is completion tokens; Gen is model-only GPU generation time in seconds. Some paths ran in separate sessions, so Gen is descriptive rather than a controlled end-to-end efficiency comparison.

	QuixBugs		**HumanEvalFix-Python**	
Method	Tok	Gen (s)	Tok	Gen (s)
Full Rewrite	211.5 [171.2,286.0]	7.50 [5.99,10.28]	178.0 [142.2,226.0]	6.15 [4.83,7.72]
Unified Diff	49.5 [45.0,61.8]	1.94 [1.83,2.31]	68.0 [60.0,75.0]	2.52 [2.27,2.78]
AdaEdit-B	59.0 [50.8,74.2]	2.30 [2.00,2.82]	60.5 [40.5,75.0]	2.29 [1.58,2.80]
AdaEdit-F	69.0 [59.0,76.0]	2.65 [2.27,2.92]	66.0 [46.2,90.8]	2.48 [1.88,3.34]
Agentless	40.5 [36.0,52.5]	1.61 [1.50,2.09]	49.5 [43.0,59.2]	1.91 [1.67,2.27]
SWE-agent	34.5 [32.0,38.0]	1.45 [1.35,1.67]	38.0 [34.0,43.0]	1.56 [1.41,1.73]
CODESTRUCT	87.5 [77.0,106.2]	3.21 [2.86,3.76]	96.0 [84.0,106.5]	3.45 [3.07,3.75]
Ours	154.5 [145.8,162.5]	5.44 [5.23,5.83]	160.5 [153.5,169.2]	5.59 [5.34,5.93]

Table 26: Post-hoc source-frame (U) and Python AST agreement on the same candidate files as Table 1; counts are unconditional over 40 attempts. The local Patch frame result follows from its applicable splice construction.

	QuixBugs		HumanEvalFix-Python	
Method	U	AST	U	AST
Full Rewrite	24	39	26	38
AdaEdit-B	0	0	0	0
AdaEdit-F	0	0	0	0
Agentless	37	37	38	38
SWE-agent	38	38	39	39
CODESTRUCT	36	36	40	33
Ours	40	40	40	40

Rewrite retains the frozen source frame on only 24 and 26 tasks, while its AST agrees with the target on 39 and 38. This separates a large source-relative lexical preservation difference from a much smaller structural difference. For CODESTRUCT's HumanEvalFix string-edit path, all 40 candidates preserve the frame but only 33 match the target AST, locating its remaining losses inside the writable region. These diagnostics reuse the original responses and were chosen after seeing the primary endpoints; they are mechanism checks rather than independent confirmation of a reliability gain.

L Address Binding and Multi-Span Boundary Studies

Proposition 5 identifies a distinction between source-version binding and insertion-boundary binding. We replay the 40 recorded 32B QuixBugs Patch responses against a controlled source perturbation that prepends 30 characters. Keeping the original source identity rejects all 40 responses. In a deliberately unsafe variant, recomputing the identity for the changed source while retaining the old numeric offsets rejects 36 but accepts four at an incorrect location; all four contain a zero-length insertion whose expected old value is ϵ. Thus the frame law holds for the *addressed* splice but does not prove that this address remains the intended one after an external source change.

A separate, post-hoc applicability variant requires 16 characters of matching source context on each side of an insertion. It accepts all 40 recorded responses on their original sources and rejects all 40 after the same rebinding perturbation. This is a controlled boundary diagnostic, not an implemented

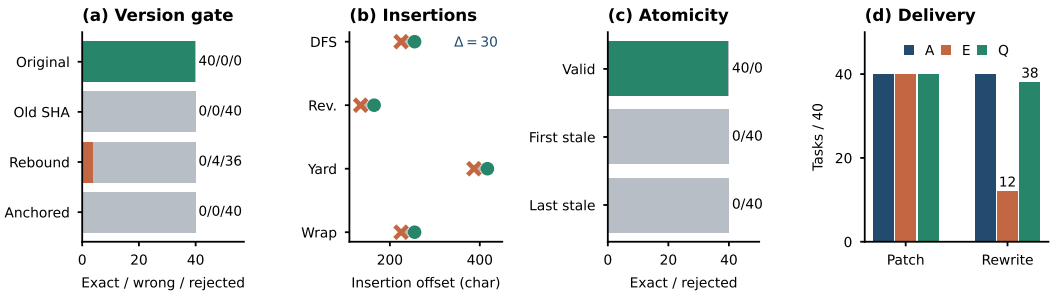


Figure 9: Boundary diagnostics ($n = 40$ per panel). (a) Exact/wrong/rejected under the original source, old hash, rebound hash, and a separate post-hoc context guard. (b) Four misbound insertions: crosses are stale offsets, circles the intended offsets after a 30-character shift. (c) Exact/rejected for valid and corrupted gold multi-span lists, without model calls. (d) Fresh 32B fixed-intent delivery: applicable A, byte-exact E, released-test pass Q.

property of the main method: repeated contexts or different changes could collide, and an unchanged original version identity remains the simpler rejection rule for an unapproved source version.

The transactional premise is checked independently with 40 gold multi-span operation lists. All valid lists construct the exact target. Corrupting the expected old text in either the first or last operation rejects every list and leaves no partial mutation. This is a deterministic evaluator test rather than a new model call; it tests atomic application conditional on the operation list, not intent discovery.

We also run a fresh one-call 32B BF16 comparison on 40 *single-file, two-or-more-span* QuixBugs tasks. Each supplied intent combines the original repair with a synthetic, nonsemantic terminal comment; 39 tasks have two spans and one has three. The comment makes unchanged and changed text jointly observable in byte-exact scoring, but the released program tests do not establish its presence. Local Patch yields $A/Q/E = 40/40/40$; full Rewrite yields $40/38/12$. The paired exact-file discordance is $28 : 0$ (two-sided exact McNemar $p \approx 7.45 \times 10^{-9}$), whereas released-test discordance is only $2 : 0$ ($p = .5$). Median completion tokens are 236.5 for Patch and 219.5 for Rewrite; median GPU generation times are 8.07 and 7.43 seconds. Hence this stress test supports exact multi-span delivery under its fixed intent, while providing no token or latency advantage. It does not establish behavior on multiple files or independently meaningful simultaneous repairs.

M Real Multi-Operation Repair Delivery

The preceding QuixBugs multi-span check added a nonsemantic comment. Here both sets instead use *released, independently meaningful multi-operation fixes*. All 12 multi-operation HumanEvalFix-Python examples in the original 164-example repair collection have failing buggy programs and canonical successors that pass their released author tests: 11 require two supplied operations and one requires four. Separately, we derive 40 Python debug cases from the CodeEditorBench Primary release (Guo et al., 2025). We require two to four nonadjacent, nonempty line replacements, a released public example failed by the buggy source and passed by the canonical successor, then take the first 40 in a fixed hash order (26 two-, 13 three-, and one four-operation cases). These are single-file cases with correct source addresses and replacement strings supplied to both interfaces; neither experiment tests intent discovery or cross-file repair.

Both studies use the pinned 32B BF16 checkpoint, greedy decoding, a 4,096-token cap, one call per task and interface, and alternating within-task call order. The compact local contract asks for one JSON replacement string per supplied operation, checked against the original source hash and old values before ordered splicing; the full contract asks for the whole corrected file. Direct deterministic application of the supplied values attains every canonical target without inference. Raw responses were replayed independently, including test execution and AST comparison. The compact contract here differs from the main-table Patch envelope, so these are separate paired studies rather than additional main-table baseline rows.

Table 27: Real multi-operation, supplied-intent delivery. A: applicable candidate; Q: released HumanEvalFix author tests or CodeEditorBench public example; E: byte equality to the single released target; AST: target structural agreement. Tok and Gen are median completion tokens and model-only GPU seconds. Each denominator is the number of tasks in that subset.

Subset	Interface	A	Q	E	AST	Tok	Gen
HumanEvalFix (12)	Full Rewrite	12	11	0	11	248	8.81
	Compact local	12	**12**	**12**	**12**	**28**	**1.27**
CodeEditorBench (40)	Full Rewrite	40	36	21	35	123.5	4.34
	Compact local	40	**40**	**40**	**40**	**64**	**2.40**

In HumanEvalFix, all 11 test-passing but byte-inexact rewrites have the target AST; the one remaining rewrite emits literal escaped newline sequences and is invalid Python. Thus $12/12$ versus $0/12$ canonical fidelity is primarily lexical, while the released-test difference is one output-encoding failure. On CodeEditorBench, 14 of the 15 public-example-passing noncanonical rewrites also have the target AST; the four public-example failures omit or reindent executable control/output statements. Paired exact-file discordances favor local delivery $12:0$ and $19:0$ (exploratory two-sided exact sign $p = .00049$ and $p = 3.8 \times 10^{-6}$); released/public-test discordances are only $1:0$ and $4:0$ ($p = 1$ and $.125$). The CodeEditorBench exact-file difference is $+47.5$ points with a descriptive 95% task-bootstrap interval $[32.5, 62.5]$. The local advantage in completion cost is consistent across all 12 HumanEvalFix task pairs but heterogeneous on CodeEditorBench (26/40 use fewer tokens); Figure 10 shows paired spreads rather than just medians. The CodeEditorBench selection and Q use a public example, so these results are *not* a score on its official benchmark or hidden test suite. In an exploratory direct-execution probe of eight private examples per selected case, even the released canonical target passed all eight in only 1/40 cases under our extracted harness; we therefore do not use that probe to rank the interfaces. These data support conditional multi-operation delivery and leave functional generalization unresolved.

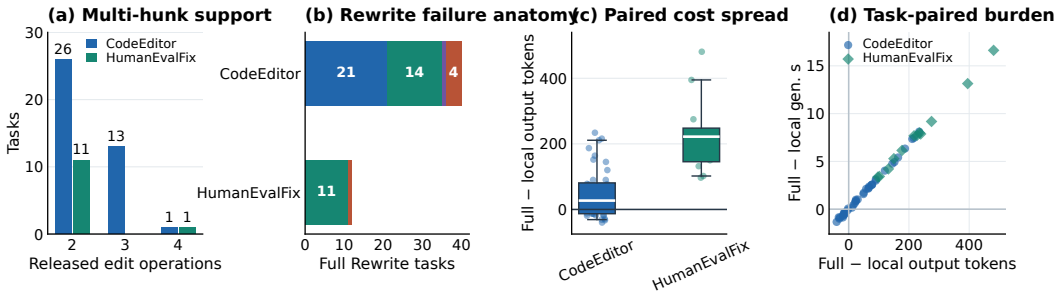


Figure 10: Diagnostics on the two real multi-operation subsets. (a) Number of released repair operations per task. (b) Full Rewrite outcomes partitioned into exact target (blue), test-passing and AST-equal but byte-inexact (teal), test-passing and AST-different (purple), and test-failing (rust). (c) Task-paired full-minus-local completion-token distributions. (d) The same token differences against model-only generation-time differences; positive values favor local delivery. Panels use distinct denominators (40 CodeEditorBench, 12 HumanEvalFix) and do not pool outcome rates.

N Supplementary Scope and Delivery Experiments

We ran five additional single-checkpoint studies designed to probe distinct assumptions of the framework. All model calls use the same pinned Qwen2.5-Coder-32B-Instruct BF16 checkpoint, greedy decoding, and one attempt per cell. Tasks and prompts were frozen before their respective formal calls. The cells below are separate studies and are not pooled into a single treatment estimate.

N.1 TRANSFER TO PUBLIC PACKAGE CONFIGURATIONS

Forty public `package.json` configurations from Babel (21), SvelteKit (7), and Vitest (12) were selected from fixed repository revisions. Each receives a controlled, type-preserving dependency-version update; these updates are constructed tasks, not historical commits. Leaf, parent-object, and full-root delivery receive the same authoritative parsed state and correct edit. Exactness means equality of the complete parsed JSON successor, not byte equality of the original file; structural package validity checks the retained package fields but does not execute project tests. All 120 responses were accepted and retained the protected source frame. Exact counts were 26/40, 40/40, and 40/40 for leaf, parent, and full. The leaf result reverses a simple smaller-output ranking: all 14 errors double-encode the replacement scalar as a string containing a second JSON string literal. After inspecting these errors, we froze an *exploratory* 40-call leaf prompt that explicitly shows the native JSON-string encoding. It gives 40/40 exact parsed states, while the original 26/40 result remains the primary observation. This diagnosis supports a representation-boundary explanation, not a universal scope ordering. The project-specific counts and response-cost tradeoff are shown in Figure 11.

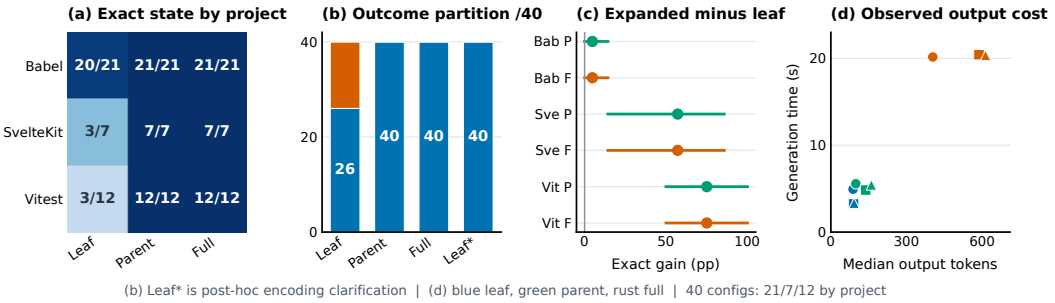


Figure 11: Public configuration transfer, 40 constructed version updates, one model call per scope and task. (a) Exact parsed-state counts by project and original scope. (b) Unconditional outcome partitions; Leaf* is a separate post-hoc encoding-clarification prompt. (c) Within-project paired parent/full minus original leaf exact-rate differences, with task-bootstrap 95% intervals. (d) Median completion tokens versus generation time; color gives scope and marker shape gives project. All 14 original leaf errors are extra string encoding.

N.2 AMBIGUOUS TEXTUAL ANCHORS

Forty held-out HumanEvalFix-Python repairs have a single correct writable span. We insert zero, one, or three nonexecuted copies of its old text into the module docstring, then compare source-bound addressing with a deliberately simple first-match text replacement control. Gold successors pass and perturbed buggy files fail the released author tests as intended. With no decoy, both arms produce 40/40 exact and test-passing repairs. With one or three decoys, source-bound addressing produces 39/40 while first-match produces 0/40; every first-match failure selects a decoy address. The control is intentionally naive and is not a reproduction of any published agent or `SEARCH/REPLACE` system. This stress test isolates why an address witness matters when text alone is ambiguous; it does not establish a ranking over complete repair methods.

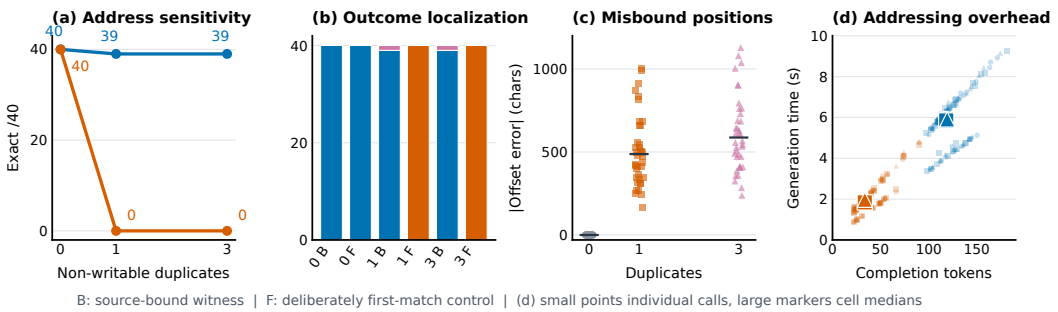


Figure 12: Address ambiguity on 40 held-out repairs per dose and arm. (a) Exactness; (b) exact, wrong-address, and rejected outcomes (B: source-bound; F: naive first match); (c) first-match absolute offset errors; (d) completion tokens versus time, with individual calls and cell medians. Marker shapes encode zero, one, and three nonexecuted docstring decoys.

N.3 A MEASURED INTENT-INFERENCE BRIDGE

Forty disjoint eligible HumanEvalFix-Python repairs test a two-stage pipeline: infer a one-span intent, then deliver the same intent through local or full-file interfaces. Three invalid intents skip both deliveries; their six cells count as failures. Of 37 valid intents, 23 match the canonical repair and 32 pass released tests. Local and full-file delivery produce 23/40 and 0/40 byte-exact files, respectively, but each passes tests on 32/40. A post-hoc audit finds 21 AST-equivalent full-file candidates, including 17 that differ only in leading blank lines; 20 of 23 canonical-intent full-file candidates are AST-equivalent. Thus the released tests show no behavioral local advantage here: lexical fidelity differs, while intent inference remains a shared bottleneck.

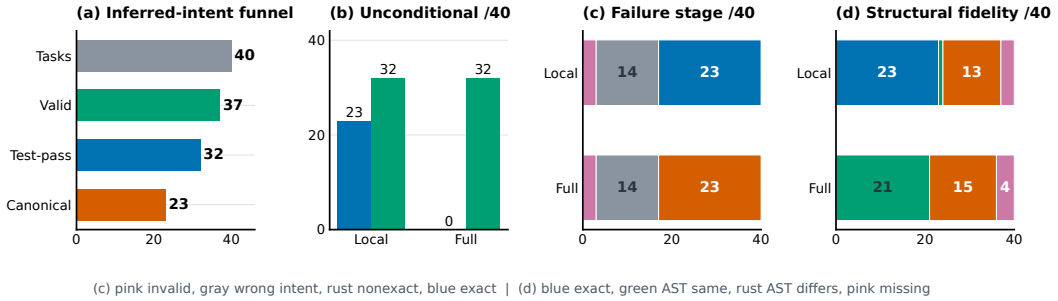


Figure 13: Two-stage held-out intent bridge, 40 tasks. (a) Inference funnel; (b) unconditional byte-exact and test-passing outcomes, counting skipped deliveries as failures; (c) failure stage; (d) post-hoc structural fidelity for both paths: byte exact, AST-equivalent but byte different, AST different, or missing. Seventeen of 21 AST-equivalent full-file candidates differ only in leading blank lines.

N.4 MATCHED-TOKEN PROTECTED CONTENT

We cross three unchanged payload types (periodic text, prose, and high-entropy base64url) with two tokenizer-matched lengths and leaf versus parent delivery on the same 40 public-JSON states. Every task has the same correct scalar edit across cells. The payload is an unchanged sibling of the edited leaf, and the model sees the complete source in both delivery paths. The 384- and 2,048-token targets are achieved to within two tokens. Token matching does not match character lengths: Table 28 gives the realized medians. The prose stimulus is a fixed, author-written passage rather than an external corpus sample. Consequently this factorial compares these constructed content classes under a fixed checkpoint, but cannot isolate a latent “copyability” mechanism from character length or lexical format.

Table 28: Realized unchanged-payload burden in the 40-task content factorial. Entries are task medians; token bins are matched by the pinned model tokenizer, whereas character and compressed-byte lengths vary by class.

	384-token bin		2,048-token bin	
Payload	Chars	zlib bytes	Chars	zlib bytes
Periodic	822.0	45.0	4,389.5	65.0
Prose	1,626.5	913.5	8,617.0	3,928.0
High entropy	521.5	427.0	2,781.5	2,127.5

All six leaf cells yield 39/40 exact successors. Parent exact counts change from 2 to 0/40 for periodic text, 24 to 9/40 for prose, and 31 to 26/40 for high-entropy content as the payload moves from 384 to 2,048 tokens. The paired scope-by-dose interactions are $+5.0$ (descriptive task-bootstrap 95% interval $[0, 12.5]$), $+37.5$ ($[17.5, 55.0]$), and $+12.5$ ($[0, 27.5]$) percentage points in the same class order. Periodic parent output hits the 4,096-token cap on 29/40 and 32/40 tasks; the corresponding prose and entropy cells have no length stops. This is the *most* compressible constructed string by zlib bytes (Table 28), yet it has the lowest parent fidelity. The result rules out a simple "more compressible means easier to reproduce" account under these prompts. It does not identify a neural copy mechanism: character count, lexical regularity, and compressibility move together across classes. Figure 4 summarizes exact counts and failure partitions.

N.5 Post-hoc diagnosis of generation length

The periodic parent failure is not explained by a gold response that exceeds the completion budget. We tokenized the canonical gold response for every original content-study cell using the pinned model tokenizer. All 480 gold responses fit the 4,096-token cap with a two-token margin. In the periodic parent cells, the median gold response requires 500 and 2,165 tokens at the two doses, while the median observed completion reaches 4,096 tokens in both. The fixed periodic motif appears more often than in the gold response in 31/40 and 33/40 raw completions, respectively; 29/40 and 32/40 completions stop at the cap. This directly records repeated expansion in many failures, including some that finish before the cap, while leaving other causes unresolved. By contrast, the 2,048-token prose parent cell has a median output-to-gold token ratio of 0.82 and no length stops. Figure 4c contrasts their median generated and feasible gold lengths. The token ratios are descriptive because valid JSON can differ in whitespace from the canonical gold serialization; the motif counts use the raw response text and are independent of JSON parsing.

N.6 Character-matched content replication

To probe the character-length confound in the token-matched factorial, we froze a separate exploratory study on the same 40 source states and intended edits. Each constructed unchanged payload contains exactly 2,000 characters; content class and leaf/parent delivery are crossed for 240 one-shot calls with the same pinned checkpoint, evaluator, and 4,096-token completion cap. The new material uses an independent deterministic seed. Matching characters does not match tokenizer burden or lexical structure (Table 29), so this study tests whether the original contrast survives one alternative size control rather than identifying a unique latent copying mechanism.

Table 29: Realized burden of the character-matched replication. All payloads have 2,000 characters; entries are medians over 40 task-specific strings.

Payload	Tokenizer tokens	zlib bytes
Periodic	935.0	54.0
Prose	455.5	1,088.5
High entropy	1,469.0	1,538.0

All 240 canonical gold responses fit the cap; the largest requires 2,649 tokens. On the complete, independently replayed ledger, each leaf class yields 39/40 exact successors. Parent exact counts are 0/40 for periodic text, 21/40 for prose, and 27/40 for high entropy. Relative to periodic text on the same 40 source tasks, the exploratory parent differences are $+52.5$ points for prose (task-bootstrap 95% interval $[37.5, 67.5]$) and $+67.5$ for high entropy ($[52.5, 82.5]$). Periodic parent output reaches the 4,096-token cap on 30/40 tasks and exceeds the gold motif count on 33/40; neither prose nor entropy has a length stop. Median parent completion length is 4,096, 605.5, and 1,592.5 tokens, respectively. Thus the periodic deficit persists when characters are held fixed, although tokenizer length, compressibility, and lexical regularity remain coupled. These contrasts are exploratory, not a causal decomposition of those properties.

N.7 Output-budget feasibility on longer native states

A disjoint set of 40 Babel configurations receives one constructed dependency-version update and an unchanged 640-token prose field. Leaf and full-root delivery are crossed with completion caps of 256, 1,024, and 4,096 tokens. The same source, intent, and parsed-state exactness evaluator are used within each task; only requested scope and output cap vary. Before model inference, a canonical gold response was tokenized against each cap. All 40 leaf responses fit each cap; zero, 30, and 40 full-root gold responses fit the three successive caps. Thus the 256-token full-root cell is structurally infeasible, and any exact-rate comparison at that cap must be read as a budget frontier rather than a model error rate. The injected field creates a controlled longer state; it is not part of the original public file.

Leaf exactness is 34/40 at all three caps, with the same six double-encoded version strings in every cell and no length stops. Full-root exactness is 0/40, 1/40, and 38/40; the first two cells have 40 and 39 length stops, respectively. At 4,096 tokens, all full responses finish within the cap, and the two nonexact candidates change the protected prose field while writing the intended dependency value. The full-root median completion grows from 256 to 1,024 to 1,106 tokens as the cap grows, versus 88 for leaf throughout; median generation time is 8.85, 33.70, and 36.27 seconds versus approximately 3.4 seconds. The high-cap exact ranking reverses the low-cap ranking because the smaller interface retains a scalar-encoding error. Figure 14 separates feasibility, truncation, quality, and cost; it does not imply that unrestricted full reconstruction is generally preferable.

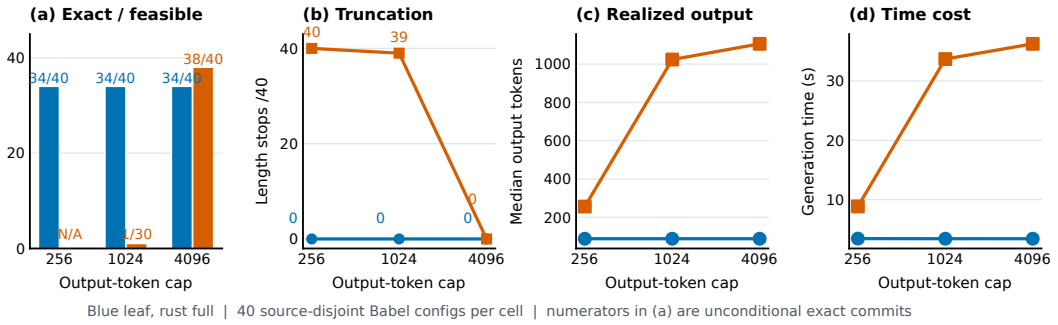


Figure 14: Output-budget frontier on 40 derived, source-disjoint public configurations per cell. Blue denotes leaf and rust full-root delivery. (a) Exact parsed-state count over gold-feasible count; N/A marks the full-root 256-token cell with no feasible gold response, and numerators remain unconditional over 40 attempts. (b) Length-stop counts. (c) Median completion tokens. (d) Median GPU generation seconds. Lower caps induce truncation for full-root output; high-cap leaf failures are string-encoding errors.

The same scalar-encoding failure also appears in this source-disjoint budget set. After inspecting the six 256-token leaf failures, we froze an exploratory prompt with the same native JSON-string example used in the earlier public-configuration probe and reran all 40 budget leaf tasks at the unchanged 256-token cap. Exact parsed-state outcomes rise from 34/40 to 40/40; all clarified outputs are accepted, and none stops at the cap. The median completion increases from 88 to 97 tokens. This cross-set replication supports an encoding-contract diagnosis for these particular leaf failures, while

remaining a post-hoc prompt intervention rather than a prespecified scope comparison. Both original 34/40 and exploratory 40/40 outcomes are retained.

O Temporal Replication Sensitivity

Table 30: Fixed-intent results across two deployment snapshots.

Run	Patch / Rewrite	Difference	95% CI
Historical reconstructed	46/48 vs. 26/48	$+41.7$ pp	$[27.1, 56.3]$
Later native-event rerun	48/48 vs. 47/48	$+2.1$ pp	$[0, 6.3]$

The two 48-task fixed-intent snapshots use aligned task conditions, yet the measured Patch-minus-Rewrite gap changes sharply. Their model alias does not identify immutable checkpoint bytes, and the later execution took place under a different client/environment configuration. This observation identifies temporal deployment sensitivity; it cannot assign the change to a particular provider revision, software component, or stochastic response process.

P Limitations and Interpretive Boundaries

The theoretical frame guarantee depends on a correct writable footprint, source-bound applicability checks, sound validation, and atomic installation. It does not guarantee a correct value inside the footprint or semantic correctness of the successor. The exposure partition describes which original positions are regenerated, not a learned neural representation or an error-rate law. The binary and JSON tasks offer controlled output-scope contrasts but do not capture unrestricted program repair. The program study supplies a correct edit intent and evaluates one-call delivery actions or format paths from published systems, so its scores are not end-to-end agent scores. The added real multi-operation cases remain single-file, and CodeEditorBench correctness uses selected public examples rather than its official score. Released tests are finite; byte equality is stricter than many behaviorally acceptable repairs and should be read alongside test pass.

The matched-scope JSON volume interaction is supported within the opaque-field experiment, yet the later structured-list transfer does not establish the same dose trend. Those studies reuse task documents and differ in payload organization and hosted snapshot. The relaxed-budget diagnostic also spans a later snapshot; it shows that a larger cap did not remove the observed deficit, while leaving a causal budget explanation unresolved. Provider aliases may change over time. We therefore interpret each paired contrast under its own task distribution, model configuration, and date rather than asserting a universal interface ranking.

The new package updates are constructed version changes on public configurations, not historical fixes; structural package checks do not substitute for building each project. The leaf encoding clarification followed an observed failure and remains exploratory. The ambiguous-address first-match control is deliberately simple. In the inferred-intent bridge, both delivery paths share the same one-shot inferred intent and tie on finite released tests, so their byte difference is not a demonstrated behavior gain. The content factorial matches tokenizer counts but not character lengths or lexical regularity; its periodic length stops do not establish a general law about compressibility. The budget study also adds a controlled prose field, and its smallest full-root cap is gold-infeasible by construction.

Q Ethics Statement

The experiments use generated bit strings, public project JSON, and public benchmark programs. They do not use private user accounts or credentials, and no generated repair was deployed to a production repository.