REVISION CONSISTENCY UNDER ONE-GAP KV REUSE: A CONTROLLED STUDY OF STRUCTURED OUTPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Selective key–value (KV) reuse after a prompt edit can leave a structured output schema-valid but revision-inconsistent. We study one-gap reuse, which recomputes the edited chunk while restoring the cached suffix. On matched 192-request stress suites, Reuse reduces exact-record accuracy from 54.17% to 45.31% for Qwen3-8B and from 47.92% to 41.15% for Llama-3.1-8B; partial recomputation is non-monotonic, whereas full-tail recomputation recovers Dense's outcomes. On 384 natural MultiWOZ revisions, the predefined exact-record difference is inconclusive, but a post-hoc changed-field endpoint shows a $-6.51\,pp$ paired difference. RevisionPatch-Exact, a frozen whole-field old→new rule, raises Reuse exact accuracy from 14.06% to 16.41%, yet seven of 51 actions modify an unrelated field and introduce an error despite zero exact-record breaks. Trusted-key replay avoids these observed harms. Together, these results establish a policy-specific revision-consistency failure and show why output repair must be evaluated below the record level.

1 INTRODUCTION

Long-context language-model applications often revise information in a shared history. A user may change a hotel preference, or an agent may update a tool argument. The resulting structured record should reflect the revised request rather than the superseded history; we refer to this requirement as *revision consistency*. Reusing cached key–value (KV) states can reduce repeated prompt computation (Kwon et al., 2023; Gim et al., 2024). However, an edit changes the context on which later states depend: even if the following text is unchanged, its cached states were computed from the earlier prompt. A generated record may consequently retain a superseded value while remaining parseable and schema-valid. Syntax validation alone cannot detect this revision-consistency failure.

Prior work has shown that downstream KV states can retain information from edited text (Li, 2026; Li et al., 2026). We examine when this retained information changes structured-record correctness and how suffix recomputation changes the outcome. We study *one-gap reuse*, which reuses the unchanged prefix, recomputes the chunk containing the edit, and restores the cached suffix. Comparing this policy with progressively longer suffix recomputation and full prompt recomputation isolates the executed cache policy under a fixed generation and evaluation procedure; natural revisions then test whether the same consistency gap appears beyond the stress construction.

Revision consistency also exposes an output-level repair signal. For an explicit value revision, an incorrect record may retain the old value as a complete field while the runtime edit supplies both old and new text. We study this signal with RevisionPatch-Exact, a deterministic rule that updates a unique whole-field old-value match when the new value is absent. The rule requires no model call and operates only on the output. Because value-level uniqueness does not identify field semantics, we audit every action at field level and find both useful corrections and wrong-field updates.

Our experiments separate the cache-policy effect from the broader repair opportunity. On matched 192-case Qwen and Llama stress suites, one-gap reuse lowers exact-record accuracy by $8.85\,pp$ and $6.77\,pp$, respectively; partial recomputation is non-monotonic, while full-tail recomputation recovers Dense's request-level outcomes. On natural MultiWOZ revisions, the predefined exact-record endpoint is inconclusive, whereas a post-hoc paired changed-field analysis shows a $6.51\,pp$ Reuse deficit. The whole-field rule fixes nine complete Reuse records, but seven of its 51 actions

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.

update an unrelated field and introduce a new field error. Binding edits to a supplied intended key avoids these actions in post-hoc replay while preserving the exact-record gain.

Our contributions are:

- We characterize revision-consistency failures under one-gap KV reuse through paired interventions on two 192-case controlled suites and a paired natural-revision study.
- We audit an explicit-edit output rule at field level: 44/51 actions target the intended field, while seven introduce an unrelated field error that exact-record fix/break counts miss.
- We separate the repair signal from field identification through key-aware replay and targeted decoy controls, and compare the resulting coverage–safety trade-off with retained-record and regeneration alternatives.

2 Problem Setting

2.1 A revised prompt and a structured record

Let $r = (r_1, \ldots, r_m)$ be the reference prompt whose KV states are cached, and let $x = (x_1, \ldots, x_n)$ be the revised target prompt. The serving runtime supplies an alignment and an edit-containing chunk. We study one-gap plans whose unchanged prefix and suffix can be retrieved under the supplied alignment.

The language model generates a structured record from x. A parser maps the output to schema fields and their values. For example, after a user changes `expensive` to `moderate`, a hotel record may still contain `hotel-pricerange=expensive`. Such a record can satisfy the schema without reflecting the revision. The task is to return a record consistent with the revised request, not merely a syntactically valid output.

2.2 Three cache policies

Dense recomputes the entire target prompt. **Reuse**, our shorthand for *one-gap reuse*, restores the reference prefix, recomputes the edit-containing chunk, and restores the reference suffix. **Full-tail** restores only the unchanged prefix and recomputes from the edit-containing chunk to the prompt's end. Full-tail therefore discards every cached suffix state that could depend on the edited text.

The distinction concerns causal history, not just token equality. Prefix states precede the edit. In contrast, a restored suffix state was computed while attending to the reference text, even when the suffix tokens themselves have not changed. We call this a *stale suffix*. Restoring it can affect subsequent generation. We measure the resulting changes in record correctness by comparing the cache policies on the same requests.

To measure this effect, the controlled experiment also recomputes progressively longer portions of the suffix. Model, tokenizer, decoding, parser, and evaluator are held fixed across policies. Section 4 specifies the chunk sizes, request lengths, and paired correctness measures.

3 Output-Level Edit Rules

We first define the frozen whole-field rule evaluated in the primary experiment, then a post-hoc key-aware variant motivated by its field-level audit. The runtime edit supplies the old and new text; the parsed record supplies candidate fields. Figure 1 summarizes the frozen rule. Neither rule trains a model, modifies the cache, or regenerates output.

3.1 Recover the runtime edit

The inputs are the reference and target token sequences, the runtime edit range and execution metadata, the parsed record, the schema, and parser/schema validity. Gold records, gold changed-field labels, and Dense or Full-tail outputs are not inputs.

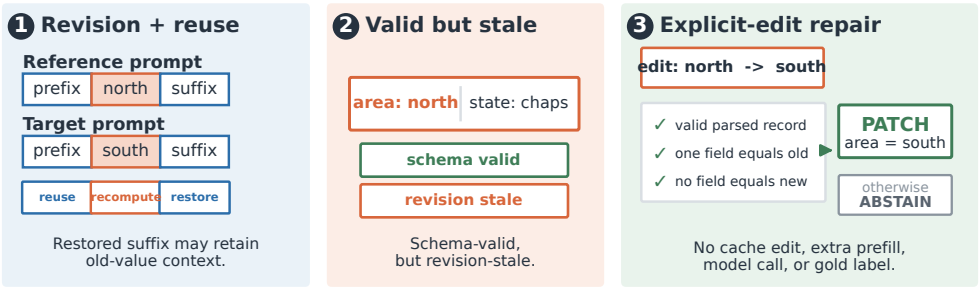

Figure 1: **Revision-consistency risk and the repair signal.** Selective reuse may restore suffix states conditioned on a superseded value, so an output can remain schema-valid while contradicting the revised request. RevisionPatch-Exact uses only the explicit runtime edit and the generated record: it replaces a unique whole-field old-value match when the new value is absent, and otherwise abstains. The rule does not modify KV states or invoke the model.

To recover the edit without dataset annotations, remove the longest common token prefix and then the longest nonoverlapping common suffix. Denote the remaining old and new text fragments by o and u. Their spans must lie inside the declared runtime edit range. The resulting fragments represent the minimal token difference between the prompts.

Field matching uses a fixed string normalizer ν to compare complete values. Matching is literal after normalization. Appendix D.1 specifies the normalization operations and their order.

3.2 Match one complete field and update it

Let K be the set of schema field keys and V_k the parsed value of field k. We need to know whether the old value occurs in exactly one field and whether the new value is already present. Define

$$M_o = \{k \in K : \nu(V_k) = \nu(o)\}, \qquad M_u = \{k \in K : \nu(V_k) = \nu(u)\}. \tag{1}$$

For valid inputs, RevisionPatch-Exact acts only when $\nu(o)$ and $\nu(u)$ are nonempty and distinct, $|M_o| = 1$, and $|M_u| = 0$.

When the condition holds, let $k^\star$ be the unique member of M_o. Replace the complete value $V_{k^\star}$ with the NFKC- and whitespace-normalized surface of u, and copy every other field unchanged. Otherwise return the original output. Writeback applies NFKC and whitespace cleanup but preserves case and punctuation; it does not write the match-normalized string.

Execution order. First validate the parsed record and edit alignment. Next recover and normalize o and u. Reject empty or identical normalized fragments. Then compute M_o and M_u using whole-field equality. Finally, replace the unique field if the condition holds; otherwise abstain. All decisions use the runtime inputs specified above.

3.3 Scope of field matching

Whole-field equality avoids modifying substrings inside names or composite values. Uniqueness avoids choosing among several old-value matches. The new-value check prevents an update when both versions already occur in the record. For example, a revised train departure may coexist with the old city as a legitimate destination.

These syntactic checks do not identify the intended field semantically. If an unrelated field uniquely equals o, the rule can modify the wrong field. It also does not repair malformed outputs or infer omitted values. We therefore evaluate both record-level transitions and changes to the complete set of erroneous fields. Matching scans the serialized field values once and requires no model call or additional prefill.

3.4 KEY-AWARE BINDING AS A POST-HOC SAFETY ANALYSIS

Suppose the runtime supplies a trusted intended key $k_e \in K$. The key-aware replay replaces V_{k_e} only when $\nu(V_{k_e}) = \nu(o)$ and $|M_u| = 0$; otherwise it abstains. This binding tests whether the repair signal can be separated from field identification. It was derived after the primary whole-field results and is reported as a safety analysis, not a preregistered method. If k_e is unavailable, the replay is not deployable from the original inputs.

4 EXPERIMENTAL SETUP

We evaluate three questions: how one-gap reuse changes record correctness; when explicit-edit output rules repair or harm generated records; and what serving cost remains under the tested configuration. Controlled requests, natural revisions, and long-context requests answer different questions and have separate denominators.

4.1 WORKLOADS AND EXECUTION

Controlled requests. The primary suite uses Qwen3-8B on 192 prompts: four suffix families crossed with four edit-count strata, with 12 examples per cluster. Each prompt has 128 tokens. Reuse restores $[0, 16)$ and $[32, 127)$, recomputes $[16, 32)$, and executes one decode-bootstrap token, for a cost of 17 tokens. Full-tail recomputes $[16, 128)$. Intermediate policies extend recomputation by $+16, +32, +48, +64, +80$ tokens. A separately frozen Llama-3.1-8B suite repeats the same 192-case grid. Appendix C gives both grids and the disclosed recovery chronology.

Natural revisions. The primary workload uses Qwen3-8B on 384 dialogue-disjoint MultiWOZ 2.2 revisions (Yang et al., 2025a; Zang et al., 2020). Selection is outcome-blind, with separate prompt-development and pilot sets. Prompts have 160 tokens and at most 96 generated tokens. We also evaluate SGD (Rastogi et al., 2020), additional Qwen, Mistral, and Llama models, and HammerBench (MadeAgents, 2026). HammerBench fixes tool selection and tests explicit argument revisions. Appendix A gives the data protocol; Table 3 lists evaluation sizes. Natural-workload prefill counts exclude decode bootstrap, unlike the controlled frontier's total recomputation counts.

Long-context requests. We expand 64 revisions with semantically related history to 8k and 16k tokens, at concurrency 1 and 4. Correctness and cost come from the same single-run cell. Additional clean-start repetitions measure serving cost separately. The timed systems adapter receives old/new strings from the request manifest and schema keys from the stored gold-record dictionary; it reconstructs no token diff on the timed path and receives no gold values or changed-field labels. This evaluation measures the patch core with supplied metadata. Establishing their runtime provenance requires the manifest builder, which is absent from the supplied bundle (Appendix E).

Arms use independent services with matched model, tokenizer, decoding, parser, and evaluator. Logs validate the retrieved and recomputed ranges; missing checks or unintended runtime fallback reject a run (Appendix B). Primary patch decisions are fixed before gold values are scored. The rule was frozen after formal GPU execution began but before formal outputs or gold-linked results were inspected.

4.2 BASELINES AND DEPLOYMENT ASSUMPTIONS

We compare direct patching with policies that select Full-tail generation. RevisionGuard-Exact uses the patch's match condition but requests Full-tail generation. RevisionGuard-EditMargin uses fixed string-similarity thresholds. Tables abbreviate these as Guard-Exact and Guard-EditMargin. The static baseline selects requests with the longest stale suffixes; random routing matches the Full-tail action count, not token cost. Appendix D also defines EditCoverage. A separate pre-decode controller has its own training, calibration, and test sets (Appendix C.1).

Reuse+Patch denotes Reuse followed by RevisionPatch-Exact; *Patch-Exact* is its table abbreviation. Prior-record direct update assumes a retained parsed record. Append-only erratum is an alternative target-generation path, not a second generation after Reuse. These deployment assumptions are compared separately.

4.3 METRICS AND PAIRED COMPARISONS

Exact-record accuracy requires strict equality of the complete schema-keyed records. MultiWOZ, SGD, and HammerBench stringify values and trim only surrounding whitespace; systems scoring does not trim it. Case, punctuation, internal whitespace, and Unicode compatibility differences are not normalized for these scores. Patch matching uses the separate normalizer ν. Changed-field accuracy scores the annotated revised field; parser failures remain in the denominator. Exact record is the predefined natural endpoint; its paired changed-field analysis was added post hoc and is always labeled accordingly. For every primary Patch action, the post-hoc safety audit also compares the complete before/after error-field sets, records whether the patched key equals the intended key, and classifies any newly introduced field error. Appendix A.3 specifies parsing and post-patch scoring.

For $A \to B$, the first and second letters in TT, TF, FT, and FF denote correctness under A and B. A fix is FT and a break is TF. With N paired requests,

$$\Delta(A, B) = \frac{N_{\mathrm{FT}} - N_{\mathrm{TF}}}{N}. \tag{2}$$

A net gain can therefore coexist with broken records. Reuse→Full-tail fixes and breaks are also called Benefit and Harm. We report exact two-sided McNemar tests (McNemar, 1947) and cluster-bootstrap intervals (Efron, 1979); Appendix A defines the units.

Serving metrics are time-to-first-token (TTFT), arrival-to-final-record latency, and completed-request throughput. Live guards include both generations on flagged requests. Offline routing costs instead count only the selected arm; Appendix F reconciles them. Patch adds no model request. Systems latency includes the common record handler, whereas the separate CPU replay measures patch overhead alone.

5 RESULTS

5.1 CONTROLLED RECOMPUTATION CHANGES CORRECTNESS NON-MONOTONICALLY

On the 192 controlled requests, Reuse obtains 87/192 correct records (45.31%), compared with Dense's 104/192 (54.17%). Dense→Reuse has TT/TF/FT/FF $= 78/26/9/79$: 26 records become incorrect and nine become correct. The net change is $-8.85\,pp$ (McNemar $p = .005988$; cluster-bootstrap 95% CI $[-12.50, -5.21]\,pp$).

Figure 2 shows the complete sampled chunk-aligned grid. The five partial extensions produce 86, 84, 83, 85, and 80 correct records; none recovers Dense's aggregate accuracy. Full-tail reaches 104/192 and is correct on the same requests as Dense. Their parsed records agree on 191/192 cases and their raw strings on 190/192 (Appendix C).

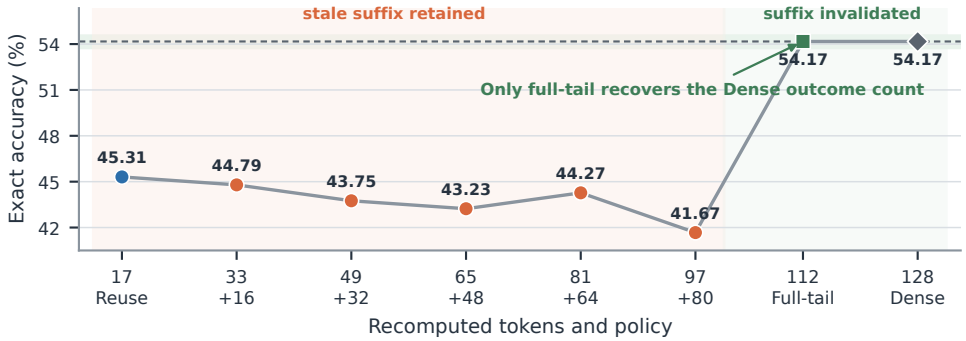

Figure 2: **Controlled recomputation frontier.** Aggregate exact-record accuracy for eight recomputation policies on the same 192 controlled requests. Extending partial suffix recomputation is non-monotonic in this grid; invalidating the full suffix recovers Dense's aggregate outcome count.

Recovery also differs across requests. Twenty cases change correctness in both directions as recomputation increases. Stable parsed-record equality to Dense starts at Reuse for 117 cases, at a partial

extension for 13, at Full-tail for 61, and is not reached for one. Thus additional recomputation is not a monotonic predictor of correctness here, and full-tail recomputation is not necessary for every individual case.

The separately frozen 192-case Llama replication gives 92 correct for Dense and Full-tail versus 79 for Reuse, a $-6.77\,pp$ paired difference (McNemar $p = .00720$; 95% CI $[-10.94, -3.13]\,pp$). All arms share the same 48 parse failures. Partial recomputation is again non-monotonic, and Full-tail matches Dense's correctness and parsed record on all cases (Appendix C).

A separate pre-decode controller uses a 640-case test set. Across its frozen settings, lift over matched-count random routing ranges from -1.05 to $+0.78\,pp$; the risk-constrained setting sends 639/640 cases to Full-tail. This controller provides little consistent gain over matched-count random routing (Appendix C.1).

5.2 Cache reuse and output repair on natural revisions

Table 1 reports 384 independent dialogues. For the predefined exact-record endpoint, Dense and Reuse yield 58 and 54 correct records; the $-1.04\,pp$ difference is inconclusive ($p = .3877$; 95% CI $[-2.86, 0.78]\,pp$). The post-hoc changed-field endpoint gives 256 versus 231 correct, a $-6.51\,pp$ difference (Holm-adjusted $p = 4.47 \times 10^{-5}$; CI $[-9.64, -3.65]\,pp$). This adjustment does not make the endpoint confirmatory; Appendix D gives the paired tables.

Table 1: Natural-revision results on 384 independent dialogues. Each arm is a separate one-pass service; changed field is a post-hoc endpoint.

Metric	Dense	Reuse	Full-tail
Exact record	58 (15.10%)	54 (14.06%)	59 (15.36%)
Changed field	256 (66.67%)	231 (60.16%)	257 (66.93%)
Parseable / nonblank	383 / 384	384 / 384	383 / 384
Mean recomputed tokens	160.00	**16.00**	73.04
P50 / P95 latency (s)	8.231 / 14.861	8.069 / 14.561	8.102 / 14.639

Applying the same patch to Dense, Reuse, and Full-tail gives 4/0, 9/0, and 4/0 exact fixes/breaks. Reuse therefore has five more fixes than either control, a $+1.30\,pp$ interaction (bootstrap 95% CI $[0.26, 2.60]\,pp$). Four of its nine fixes are Dense-correct/Reuse-wrong/Patch-correct. The remaining fixes, together with those in Dense outputs, show that the repair opportunity is not specific to stale KV. We therefore interpret reuse as one source of revision errors and the explicit edit as a broader output-level repair signal.

Table 2: Natural-revision method comparison. FT denotes Full-tail selection and P a field patch. Prefill is the offline selected-arm mean: a selected Full-tail replaces Reuse in this accounting. Actual two-pass totals include both requests (Appendix F).

Policy	Action	Exact	Δ reuse	Random lift	Prefill
Always reuse	0	54 (14.06%)	$0.00\,pp$	–	16.00
Always full-tail	384 FT	59 (15.36%)	$+1.30\,pp$	–	73.04
Guard-Exact	51 FT	59 (15.36%)	$+1.30\,pp$	$+1.13\,pp$	23.91
Guard-EditMargin	54 FT	61 (15.89%)	$+1.82\,pp$	$+1.64\,pp$	24.44
Static top 25%	96 FT	55 (14.32%)	$+0.26\,pp$	$-0.06\,pp$	35.92
Patch-Exact	51 P	**63 (16.41%)**	$+2.34\,pp$	–	**16.00**

On Reuse outputs, Patch-Exact changes 51 records and increases exact correctness from 54 to 63 ($+2.34\,pp$; nine fixes and zero exact-record breaks; McNemar $p = .003906$; Bonferroni-adjusted $p = .0195$; bootstrap 95% CI $[1.04, 3.91]\,pp$). Changed-field correctness rises from 231 to 269. These aggregate endpoints are not field-safety metrics: the audit below finds seven wrong-field updates. One patched record is correct when both Reuse and Full-tail are incorrect; a selector between those two outputs cannot produce it. Guard-EditMargin reaches 61 correct records but generates again on flagged requests.

5.3 FIELD-LEVEL AUDIT AND TARGETED CONTROLS

The whole-field rule acts on 51/384 outputs. Only 44 actions modify the intended field; seven modify an unrelated field and introduce one field error each. The 51 actions comprise nine exact fixes, 29 partial fixes that remove an error without making the record exact, six neutral actions, and seven harmful actions. Among the 13 actions that do not improve the annotated changed field, six write the intended key with a surface that differs from strict gold, while seven write the wrong key. Thus zero exact-record breaks does not imply field-level safety.

In post-hoc replay, binding to the intended key acts 45 times, preserves 63 exact records, raises changed-field correctness from 231 to 270, and introduces no field error. It avoids all seven wrong-key actions and adds one update that global uniqueness had rejected. This comparison assumes a trusted runtime key and is not a preregistered result. On 51 wrong-field-decoy controls, the whole-field rule acts incorrectly in every case, whereas key-aware binding abstains; both rules abstain on 51 target-field-missing controls because the fixed-schema validator rejects them. These deterministic controls isolate the field-identification failure without new model inference.

A separate no-revision control produces zero actions on 384 dialogues. Earlier 768-record guard ablations use only exact-record and annotated-field transitions; they are retained as coverage diagnostics, not field-safety evidence (Appendix E.3).

5.4 TRANSFER WITH THE UNCHANGED RULE

Table 3 applies the unchanged rule to other models and workloads. Exact fix/break counts are positive in every cell but can miss wrong-field harm. An SGD audit finds four wrong-field updates among 55 actions, with no new error because each target was already incorrect. Cells with two to four exact fixes remain limited transfer observations.

Table 3: Cross-model and cross-benchmark transfer. F/B denotes exact- or annotated-field transitions, not full-field harm. † marks author-authorized held-out evaluation after a negative pilot gate.

Model	Benchmark / split	N	Exact F/B	Field F/B
Qwen3-8B	MultiWOZ	384	9/0	38/0
Qwen3-8B	SGD development	384	27/0	48/0
Qwen3-8B	SGD official test	127	5/0	9/0
Qwen2.5-32B	MultiWOZ	384	2/0	7/0
Mistral-Small-24B†	MultiWOZ	384	3/0	13/0
Llama-3.3-70B†	MultiWOZ	384	4/0	15/0
Qwen3-8B†	HammerBench	384	7/0	10/0
Qwen2.5-32B†	HammerBench	384	4/0	8/0

HammerBench changes the output task from dialogue state to tool arguments. Its frozen 32-case pilot failed the preregistered gate; the 384-case runs are post-hoc author-authorized held-out evaluations, not confirmatory formal runs. Reuse is parseable on only 203/384 and 206/384 cases, with all cases retained in the denominator. The Reuse-only Mistral and Llama extensions are reported separately in Appendix E because Dense and Full-tail arms are absent.

5.5 DEPLOYMENT ALTERNATIVES TRADE COVERAGE FOR DIFFERENT REQUIREMENTS

Prior-record direct update reaches 89 correct records and append-only erratum generation reaches 95; each also breaks six Reuse-correct records. When a trusted intended key is available, key-aware binding dominates the whole-field rule on the audited endpoints. A system with a retained parsed record can instead update that state directly. Appendix D gives the post-hoc protocol and measured generation times.

5.6 DESCRIPTIVE STEADY-STATE COST OBSERVATION

Under a prebuilt reference cache with supplied old/new and schema metadata, the 16k/C1 TTFT, final-latency, and throughput ratios are $5.85\times$, $1.51\times$, and $1.61\times$ in favor of Reuse+Patch under

Table 4: Deployment alternatives on the 384 primary cases. Exact fix/break is relative to Reuse and can miss field harm. Key-aware is a post-hoc replay.

Method	Exact	Fix/break	Work / required state
Reuse	54	–	Target generation
Whole-field Patch	63	9/0	0 calls; 7 field-harm actions
Key-aware Patch	63	9/0	Trusted intended key; 0 field-harm actions
Prior-record direct update	89	41/6	0 if parsed state is retained
Append-only erratum	95	47/6	Correction generation; median $+42$ prompt tokens

their respective ratio definitions. Cache construction and metadata derivation are excluded, 8k/C4 throughput is $0.82\times$ Dense, and separate cost-only repeats worsen at high concurrency. These measurements show that the patch core itself is inexpensive in the tested steady state; they are not an end-to-end deployment result. The full figure, correctness decomposition, repeated cost diagnostics, and CPU replay appear in Appendix E.5.

6 Related Work

KV-cache reuse. Exact-prefix systems reuse computation with the same causal history (Kwon et al., 2023; Gim et al., 2024). CacheBlend, Cache-Craft, EPIC, KVLink, CacheSlide, A^3, SemShareKV, and ContextPilot explore non-prefix, compositional, or similarity-based reuse (Yao et al., 2025; Agarwal et al., 2025; Hu et al., 2025; Yang et al., 2025b; Liu et al., 2026; Zhou et al., 2025; Zhao & Mastorakis, 2025; Jiang et al., 2026); SCBench evaluates methods across the KV-cache lifecycle (Li et al., 2025). We instead measure paired record correctness after an interior prompt revision under one-gap reuse.

Editing cached information and generated outputs. *Models Take Notes at Prefill* updates information retained in downstream KV states through suffix recomputation or an erratum (Li, 2026). Other work learns cache transformations, studies budgeted contiguous recomputation, mutates caches for agentic edits, or examines retained state after logical rollback (Li et al., 2026; Mao et al., 2026; Ma et al., 2026; Zhang & Yang, 2026). StepCache verifies and selectively regenerates output steps (Nouri, 2026). We study structured-record revision consistency and field-level repair harm; our output rule does not transform the KV cache.

Structured generation and retained state. XGrammar and JSONSchemaBench address syntactic constraints (Dong et al., 2024; Geng et al., 2025), whereas a schema-valid value can still miss a revision. Dialogue-state trackers generate or update retained state (Wu et al., 2019; Kim et al., 2020; Heck et al., 2020). RevisionPatch-Exact instead receives a generated record and a runtime edit; our audit shows why literal value matching must be separated from field identification. Appendix Table 21 compares these requirements.

7 Discussion and Limitations

Scope of the cache intervention. Both controlled suites identify a revision-consistency failure of the tested one-gap policy. On natural MultiWOZ, the predefined exact endpoint is inconclusive while the post-hoc changed-field endpoint shows a paired loss. This strengthens workload relevance without estimating prevalence in other planners or production traffic. Exact-prefix caching and policies that correct all affected suffix states remain outside the tested condition.

Scope of the patch. The whole-field rule requires explicit old and new text but does not infer the intended field or make stale KV states consistent. Its exact-record gain therefore coexists with seven wrong-field updates. Trusted key binding removes these harms in post-hoc replay, but changes the runtime-input assumption. Retained-record updates and erratum generation cover more cases but require different state or computation.

Safety and statistical evidence. Of 51 primary actions, seven introduce an unrelated field error (Wilson 95% CI: 6.81–25.72%) without turning an exact-correct record incorrect, exposing the blindness of exact breaks inside already incorrect records. SGD likewise has four wrong-field actions, but no new errors because those fields were already wrong. Decoy controls establish a failure condition, not its prevalence. Consequential deployment requires trusted field binding and application-specific validation.

Reporting and cost limits. The natural selection funnel measures benchmark eligibility, not deployed exposure. Serving gains vary with concurrency, and CPU replay is not an end-to-end generation trial. The joint plot uses single-run correctness cells, supplied metadata, and a prebuilt reference cache; the repeats omit correctness. The added evidence bundle closes the reported field audit and result indices, but reproducing model generation still requires the documented serving environment.

8 Conclusion

Across matched Qwen and Llama stress suites, one-gap reuse reduces exact-record accuracy; partial recomputation is non-monotonic, whereas full-tail recomputation recovers Dense's outcomes. Natural MultiWOZ does not resolve the predefined exact-record effect, although its post-hoc changed-field endpoint shows a paired Reuse deficit. Together, these results identify a policy-specific revision-consistency failure that schema validation can miss. Whole-field editing exposes a useful repair signal, but its wrong-field actions show that repair must be evaluated at field level. Reliable deployment therefore requires validated field binding or retained structured state, rather than record-level fix/break counts alone.

AI use statement

Generative AI tools assisted literature search, manuscript editing, experiment scripting, and cluster orchestration. Formal protocols and case manifests were frozen before the corresponding model outputs were read. Model outputs and measurements came from the specified inference services; AI tools did not supply gold labels or fabricate measurements. The authors inspected the code, manifests, runtime logs, raw outputs, and analyses, verified all citations and AI-assisted changes, and take full responsibility for the work.

Ethics statement

This work uses public task-oriented dialogue and tool-use datasets and does not involve new human-subject data collection. HammerBench includes generated and manually checked examples, so it is not presented as natural user traffic. The whole-field rule introduces an unrelated field error in 7/51 primary actions even though no previously exact record becomes incorrect. We therefore do not recommend it as a field-safe update. Consequential deployment requires a trusted edit-to-field binding, domain-specific validation, and an independent record validator.

Reproducibility statement

Section 4 and the appendices specify the protocols and runtime checks. The supplementary materials provide case-level matrices, all three natural-workload arms, field audits, controls, evaluator code, hashes, and model-free reconstruction scripts. They also include recovered Full-tail logs, input manifests, serving overlays, runtime ports, and deployment inventories. Model weights and the vendor container remain external dependencies; one recovered natural-context engine differs from its recorded source hash. The artifact README documents these limits; byte-identical model reexecution is not claimed.

References

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing chunk-caches

for efficient retrieval-augmented generation. *arXiv preprint arXiv:2502.15734*, 2025. URL https://arxiv.org/abs/2502.15734.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024. URL https://arxiv.org/abs/2411.15100.

Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979. doi: 10.1214/aos/1176344552.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Generating structured outputs from language models: Benchmark and studies. *arXiv preprint arXiv:2501.10868*, 2025. URL https://arxiv.org/abs/2501.10868.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024.

Michael Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauser, Hsien-Chin Lin, Marco Moresi, and Milica Gašić. TripPy: A triple copy strategy for value independent neural dialog state tracking. In *Proceedings of the 21st Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 35–44. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.sigdial-1.4. URL https://aclanthology.org/2020.sigdial-1.4/.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

Yinsicheng Jiang, Yeqi Huang, Liang Cheng, Cheng Deng, Xuan Sun, and Luo Mai. ContextPilot: Fast long-context inference via context reuse. In *Proceedings of Machine Learning and Systems*, volume 8, 2026.

Sungdong Kim, Sohee Yang, Gyuwan Kim, and Sang-Woo Lee. Efficient dialogue state tracking by selectively overwriting memory. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 567–582. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.53. URL https://aclanthology.org/2020.acl-main.53/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Bojie Li. Models take notes at prefill: KV cache can be editable and composable. *arXiv preprint arXiv:2606.17107*, 2026. URL https://arxiv.org/abs/2606.17107.

Mufei Li, Shikun Liu, Dongqi Fu, Haoyu Wang, Yinglong Xia, Hong Li, Hong Yan, and Pan Li. KVEraser: Learning to steer KV cache for efficient localized context erasing. *arXiv preprint arXiv:2606.17034*, 2026. URL https://arxiv.org/abs/2606.17034.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a540b17fb2295c736d5afd6c507acf66-Abstract-Conference.html.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross position-aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies*, pp. 83–99. USENIX Association, 2026. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

Bole Ma, Jan Eitzinger, and Harald Koestler. Leyline: KV cache directives for agentic inference. *arXiv preprint arXiv:2606.01065*, 2026. URL https://arxiv.org/abs/2606.01065.

MadeAgents. HammerBench: Multi-turn tool-use benchmark dataset. Hugging Face dataset, Apache-2.0, 2026. URL https://huggingface.co/datasets/MadeAgents/HammerBench. Pinned commit 18b4f4ea47e8b367006391951cf7e69cefa48c73.

Mingyang Mao, Wyatt Mackey, and Xiaomin Lin. Contiguity, not importance: Budgeted repair of stale KV caches after document edits. *arXiv preprint arXiv:2609.17983*, 2026. URL https://arxiv.org/abs/2609.17983.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

Azam Nouri. StepCache: Step-level reuse with lightweight verification and selective patching for LLM serving. *arXiv preprint arXiv:2603.28795*, 2026. URL https://arxiv.org/abs/2603.28795.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8689–8696, 2020. doi: 10.1609/aaai.v34i05.6394. URL https://ojs.aaai.org/index.php/AAAI/article/view/6394.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 808–819. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1078. URL https://aclanthology.org/P19-1078/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-4461.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098.

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. MultiWOZ 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pp. 109–117. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.nlp4convai-1.13. URL https://aclanthology.org/2020.nlp4convai-1.13/.

Guijia Zhang and Harry Yang. Aborted but not forgotten: KV-cache retention breaks rollback consistency in language agents. *arXiv preprint arXiv:2608.15939*, 2026. URL https://arxiv.org/abs/2608.15939.

Xinye Zhao and Spyridon Mastorakis. SemShareKV: Efficient KVCache sharing for semantically similar prompts via token-level LSH matching. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 432–445. Asian Federation of Natural Language Processing and Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-ijcnlp.25. URL https://aclanthology.org/2025.findings-ijcnlp.25/.

Yuechi Zhou, Yi Su, Jianxin Zhang, Juntao Li, Qingrong Xia, Zhefeng Wang, Xinyu Duan, and Baoxing Huai. A^3: Attention-aware accurate KV cache fusion for fast large language model serving. *arXiv preprint arXiv:2511.17560*, 2025. URL https://arxiv.org/abs/2511.17560.

A Data Construction and Statistical Analysis

This appendix describes the primary natural-revision protocol and identifies post-hoc comparisons separately. The natural evaluation uses disjoint dialogue clusters. Every reported formal method decision was made from runtime-observable inputs before its gold record was read. The primary natural-workload model is Qwen3-8B (Yang et al., 2025a), run with temperature zero, a 160-token prompt, and a maximum of 96 generated tokens. Dense, reuse, and full-tail outputs were produced by independent tensor-parallel-2 services.

A.1 Natural-workload funnel and protocol roles

The source is the official MultiWOZ 2.2 train/dev release (Zang et al., 2020), at commit `fe0c8e65cfcd8462bd33c86e35f21addc84ca82b`. Eligibility was decided without model outcomes: a dialogue had to contain a natural revision for which a reference and target prompt could be constructed and tokenized under the declared local-replacement execution. Table 5 gives every denominator. The v2 protocol excludes all 544 dialogue IDs assigned to the earlier v1 pilot or formal manifest; it is an independent protocol, not a subset selected after viewing v1 outcomes.

Table 5: Natural-workload inventory and protocol subsets. Rows include different branches of the selection process and should not all be interpreted as consecutive filters.

Stage or frozen role	Dialogues	Revisions/cases
Official train/dev input	9,437	—
Outcome-blind eligible inventory	2,485	3,088
Compatible with the original variable-plan runtime	1,098	—
After excluding 544 earlier v1 dialogue IDs	1,941	—
Schema-compatible v2 universe	528	528
Prompt development	24	24
Engineering pilot	32	32
Untouched formal evaluation	384	384

An additional outcome-blind audit of the benchmark counts 64,150 user turns: 2,999 turns (4.67%) contain at least one literal-grounded eligible revision, comprising 3,088 slot events. Of the 1,941 eligible dialogue clusters remaining after the prior-study exclusion, 528 (27.20%) fit both frozen schema/prompt variants; the policy executes all selected compatible cases. Among 384 executed formal requests, eight (2.08%; exact Clopper–Pearson 95% CI $[0.90, 4.06]\%$) are Dense-correct/Reuse-wrong and four move in reverse. These are benchmark-trace conditional denominators, not production traffic prevalence.

There is at most one case per dialogue in the prompt-development, pilot, and formal roles. Because the formal data contain only 11 reuse/full-tail exact-record discordances, below the preregistered gate of 30, training a natural-workload learned verifier was not authorized. This gate does not apply to the separate controlled pre-decode experiment in Appendix C.1.

Two terminal prompt variants were frozen before v2 output existed. Selection used only Dense parseability, then nonblank count, then a fixed variant order; correctness and changed-field accuracy were forbidden. This selected `tail_no_think`. The 32-dialogue pilot then had to pass a binding three-arm join, nonblank, parser, execution-proof, alignment, zero-fallback, and Dense/full-tail correctness-parity gate before formal launch. Pilot accuracy was not a permitted tuning signal for the method.

A.2 Three executed arms

For a target prompt with edit-containing interval $[p, g)$ and length $L = 160$, the arms were:

- **Dense**: compute all target-prompt states from scratch.
- **Reuse**: retrieve the exact prefix and the reference-prompt suffix, recomputing only the 16-token chunk $[p, g)$ that contains the edit. Thus the post-edit suffix is stale by construction.

- **Full-tail**: retrieve only the exact prefix and recompute $[p, L)$, beginning at the edit chunk. This invalidates the stale suffix but does not guarantee a correct model output.

The formal aggregate recomputation counts in Table 10 come from executed plans, not nominal ratios. The mean is 160 tokens for Dense, 16 for Reuse, and 73.04 for Full-tail. These columns count recomputed prefill only; the execution checks separately require a one-token decode bootstrap. The controlled frontier counts that bootstrap in its 17-token Reuse cost. Neither use of decode bootstrap refers to statistical bootstrap resampling.

A.3 Endpoints and inference

The MultiWOZ, SGD, and HammerBench three-arm parsers search the output for JSON-object candidates and choose the last object whose key set exactly matches the requested schema. Values are converted to strings and surrounding whitespace is stripped. Gold values receive the same string conversion and outer stripping. Exact correctness is Python dictionary equality: key order is irrelevant, but keys and every value must agree. No case folding, punctuation removal, Unicode normalization, internal-whitespace collapse, or ontology mapping is applied to these scores.

After patching, the selected replacement is compared directly with the same gold dictionary; the scorer does not apply the match normalizer ν to either record. Thus matching two values does not imply that the evaluator regards them as equal. For example, `Sunday` and `sunday` can match under ν but remain unequal for exact correctness. Changed-field scoring compares the parsed value at the annotated revised field with the stored new value. A parser failure is incorrect and remains in the denominator; the changed-field label is not an input to the core patch rule.

The systems parser also selects the last schema-matching object, but stringifies without outer stripping. Its before/after records are compared directly with stringified gold values. The controlled protocol uses strict string-record equality as reported by the authors; its runner delegates parsing to the paired-arm runtime module. These distinct evaluator conventions are preserved rather than harmonized retrospectively.

SGD is fixed at commit `e852981ae34990f4358979625854259302feaa78`; the supplied manifests identify 384 development and 127 official-test cases. HammerBench is fixed at commit `18b4f4ea47e8b367006391951cf7e69cefa48c73`, file `data/en/multi-turn.json`. Its frozen 32-case pilot did not pass the preregistered formal gate (`formal_open=false`). The later 384-case manifests were executed through an explicit author override and are therefore labeled post-hoc author-authorized held-out evaluations, not preregistered confirmatory runs.

For paired outcomes $A \to B$, TT/TF/FT/FF use the order and definitions in Section 4. We report the exact two-sided McNemar test (McNemar, 1947). If b and c are the two discordant counts, the reported value is

$$p_{\mathrm{McN}} = \min\left\{1, 2\sum_{k=0}^{\min(b,c)} \binom{b+c}{k} 2^{-(b+c)}\right\}. \tag{3}$$

Confidence intervals are percentile intervals from at least 10,000 dialogue-cluster bootstrap resamples (Efron, 1979). The controlled suite has 16 frozen `suffix-family×edit-count` clusters; we additionally enumerate its cluster sign flips. The natural formal set has one case per dialogue, but we retain the dialogue as the resampling and permutation unit to make the intended independence assumption explicit.

Matched-count random routing selects exactly the method's number B of Full-tail actions among N cases. Here B denotes the action budget. Its reported distribution uses at least 10,000 trials and a frozen seed list. This matches action count, not physical cost: a full-tail begins at the case-specific edit, so its recomputed-token count varies. We therefore report realized tokens separately for every live policy. The analytic same-count accuracy expectation is

$$\mathbb{E}[A_{\mathrm{rand}}(B)] = A_{\mathrm{reuse}} + \frac{B}{N}\left(A_{\mathrm{fulltail}} - A_{\mathrm{reuse}}\right), \tag{4}$$

Here A_{reuse} and A_{fulltail} are accuracies of the two complete policies. This expectation is a cross-check, not a substitute for the repeated random baseline.

B EXECUTION VALIDATION

Selective cache reuse was accepted only when the serving log contained eight request-joined marker classes with ranges agreeing with the manifest. Table 6 summarizes the fail-closed checks. Tensor-parallel worker markers had to appear once per rank. Any missing class, inconsistent range, duplicate case, or occurrence of `fallback=True` aborted aggregation.

Table 6: Log checks required for every selective request. These validate the executed token ranges, not the correctness of the generated record.

Marker	Required evidence
`reference_lookup`	Full reference hit, scheduler coverage, retrieved ranges, and intended recompute interval
`materialization`	Physically materialized ranges, stale-suffix and bootstrap ranges, and no fallback
`worker_load`	Rank-replicated physical token loads and scheduler coverage
`bootstrap_metadata`	Decode-bootstrap phase metadata, with no physical reload or fallback
`worker_bootstrap`	Rank-replicated bootstrap range and prior materialization
`gap_execution`	Physical model execution of the declared interior gap and scheduled-token count
`bootstrap_execution`	Physical execution of the one-token decode bootstrap
`bootstrap_plan`	External selective plan, bootstrap-only flush, and zero extra prefill at bootstrap

The global formal gate required all 384 case IDs exactly once in every arm; equality of manifest IDs, prompts, and gold records across the join; nonblank output; parser execution; valid reference/target alignment; complete proof; zero fallback; and matching model, tokenizer, prompt variant, and code version. All 384 cases passed the three-arm join. Nonblank output was 384/384 for every arm; parseability was 383/384 Dense, 384/384 reuse, and 383/384 full-tail. Every selective request had complete proof and `fallback=false`.

Archived execution evidence. The supplementary materials include all four recovered Full-tail shards, whose original hashes match the prior three-arm analysis, together with serving logs, frozen manifests, runtime overlays, and deployment inventories. One recovered natural-context engine differs from the source hash recorded in the historical job; this discrepancy is documented and the assertion is retained. The archive supports reconstruction of the recorded outcomes, while byte-identical reexecution additionally requires resolving that source version and obtaining the external vendor image and model files.

C CONTROLLED RECOMPUTATION RESULTS

The primary controlled suite uses Qwen/Qwen3-8B with tensor parallelism 2. Its 192 cases form 16 suffix-family by edit-count clusters with 12 cases each. Prompts have 128 tokens, an exact prefix $[0, 16)$, an edit chunk $[16, 32)$, and a decode bootstrap $[127, 128)$. One-gap reuse retains the stale suffix $[32, 127)$. Table 7 reports every chunk-aligned downstream endpoint. No partial extension has positive net gain over reuse, and correctness is non-monotone for 20 cases.

Table 7: Controlled downstream-recomputation frontier on the same 192 cases. Cost includes the one-token bootstrap.

Action	Recomputed	Correct	Fixed/broken	Net vs. reuse
Reuse	17	87/192	0/0	0.00 pp
+16 tokens	33	86/192	1/2	−0.52 pp
+32 tokens	49	84/192	1/4	−1.56 pp
+48 tokens	65	83/192	1/5	−2.08 pp
+64 tokens	81	85/192	2/4	−1.04 pp
+80 tokens	97	80/192	5/12	−3.65 pp
Full-tail	112	104/192	26/9	+8.85 pp
Dense	128	104/192	—	+8.85 pp

Dense→reuse has TT/TF/FT/FF $= 78/26/9/79$, a -8.85 ppchange (exact McNemar $p = 0.005988$; cluster-bootstrap 95% CI $[-12.50, -5.21]$ pp; exact cluster sign-flip $p = 0.001953$). Full-tail recomputation matches Dense's correct/incorrect status on all 192 cases. Dense and full-tail parsed records agree on 191/192 and raw strings on 190/192; the remaining differences are jointly incorrect. The earliest sampled endpoint with stable parsed-record equality to Dense is Reuse for 117 cases, +16 for 2, +32 for 1, +64 for 1, +80 for 9, full-tail for 61, and unresolved for one. Thus complete invalidation is the only sampled policy with Dense marginal accuracy, but it is not necessary for every individual case. These results support the intervention claim for this stress distribution, not a natural-workload prevalence claim.

The archived primary reports identify the model through acquisition, model-manifest, and tokenizer SHA-256 digests. The tokenizer digest is `aeb13307a71acd8fe81861d94ad54ab689df773318809eed3cbe794b4492dae4`. The recorded Git identifier is not treated as an independently documented upstream Hub revision. Complete digests remain in the accompanying verification reports.

The second-model protocol was frozen before the reported 192-case frontier outputs were generated; it discloses and excludes an earlier 64-case pilot. Four suffix families crossed with four edit-count strata and 12 repeats yield 192 cases. Table 8 reports all arms. All use Llama-3.1-8B-Instruct, TP2, and the same tokenizer-validated layouts. Forty-eight outputs are schema-unparseable in every arm and count as incorrect; the identical cases fail in every arm, so parser coverage does not explain the paired differences. The archived provenance records startup failures and engineering-only retries; the formal matrix uses one prespecified completed report per action without changing cases, prompts, decoding, endpoints, or analysis.

Table 8: Frozen 192-case Llama-3.1-8B controlled replication. Parse failures remain in the denominator.

Action	Correct	Parseable	Net vs. reuse
Dense	92/192	144/192	+13
Reuse	79/192	144/192	0
+16 tokens	79/192	144/192	0
+32 tokens	79/192	144/192	0
+48 tokens	79/192	144/192	0
+64 tokens	79/192	144/192	0
+80 tokens	74/192	144/192	−5
Full-tail	92/192	144/192	+13

Dense→Reuse has TT/TF/FT/FF $= 75/17/4/96$, a $-6.7708\,pp$ difference (exact McNemar $p = 0.007197$; 16-cluster bootstrap 95% CI $[-10.9375, -3.1250]\,pp$; exact cluster sign-flip $p = 0.0078125$). Six cases are correctness-non-monotone. Full-tail matches Dense's correct/incorrect status and parsed record on all 192 cases. This result independently replicates the controlled loss for the tested Llama suite; it does not establish the effect for arbitrary models or cache planners.

C.1 SEPARATE PRE-DECODE CONTROLLER EXPERIMENT

This controller is an auxiliary routing baseline, not part of RevisionPatch and not the 192-case recomputation suite. It uses 640 training, 320 calibration, and 640 held-out test examples. Four source clusters define train-only cross-validation folds. Logistic regression is selected over histogram gradient boosting by the sum of out-of-fold Benefit and Harm Brier scores. The selected models are refit on training data; Platt calibration and routing thresholds are then fitted on the calibration split. Test outputs are not read during these choices.

The input has 13 numerical features: edit count, changed-field count, prefix length, gap length, total length, stale-suffix length, normalized edit position and gap end, reuse ratio, old/new character similarity and Jaccard similarity, mean changed-value length, and its mean change. The terminal instruction is represented by 64 hashed character 3–5-gram features. Gold records, model outputs, source-cluster labels, suffix-family labels, trap type, and repeat index are forbidden features. Source clusters define training folds, not prediction inputs.

The controller predicts calibrated probabilities q_+ of Benefit and q_- of Harm. It chooses Full-tail when $q_+ - q_- - \lambda c$ exceeds a frozen threshold, where c is the additional recomputed-token cost. The calibration budget settings cap Full-tail call rate; their test call fractions can differ. Table 9 shows both calibration and test counts rather than interpreting the labels as achieved test budgets.

Table 9: Frozen controller test and separate matched-random diagnostics. FT counts are actual calls. The last two columns are accuracy differences in percentage points relative to 20,000 random selections: fixed-count (FC), and source-cluster/recompute-cost-stratified (SC). Budget labels constrain calibration, not test call rates. The oracle uses correctness labels and is not implementable at inference.

Policy	Cal. FT	Test FT	Exact	FC lift	SC lift
Always Reuse	–	0	240/640	–	–
Calibration budget 25%	80/320	151	252/640	-0.04	$+0.35$
Calibration budget 50%	152/320	354	262/640	-1.05	$+0.15$
Calibration budget 75%	219/320	456	282/640	$+0.78$	$+0.36$
Max-net-gain	252/320	572	288/640	$+0.24$	$+0.37$
Risk-constrained 5%	320/320	639	292/640	$+0.01$	$+0.02$
Always Full-tail	–	640	292/640	–	–
Binary oracle	–	77	317/640	–	–

The risk-constrained setting limits the implemented Wilson upper bound on residual Benefits among accepted-Reuse calibration requests to 5%. Here a residual Benefit is a request for which Reuse is wrong but Full-tail would be correct, not any model error. The code uses $z = 1.959963984540054$ and returns zero when the accepted-Reuse set is empty. We report this implemented criterion rather than call it a 5% test-error guarantee. Its calibration solution routes all 320 requests to Full-tail, and its frozen test policy routes 639/640. With only one accepted test Reuse request, the archived test bound is 0.79345 despite observing no residual Benefit.

The fixed-count random diagnostic draws 20,000 selections at each observed test action count; a second diagnostic also matches source-cluster/recompute-cost strata. Neither diagnostic changes the frozen controller. Fixed-count lift ranges from -1.05 to $+0.78\,pp$, and the stratified lifts are small and positive. These results support a limited conclusion about this controller: it gives little consistent advantage over matched routing, and the selected risk setting uses almost no Reuse. They do not prove that all pre-decode routing methods fail.

D NATURAL-REVISION RESULTS AND COMPARISON RULES

Table 10 gives the complete three-arm formal summary. For the predefined exact-record endpoint, Dense→Reuse TT/TF/FT/FF is $50/8/4/322$, giving $-1.04\,pp$ (McNemar $p = 0.3877$; dialogue-bootstrap 95% CI $[-2.86, 0.78]\,pp$). For the post-hoc changed-field endpoint, TT/TF/FT/FF is $226/30/5/123$, giving $-6.51\,pp$ (McNemar $p = 2.236 \times 10^{-5}$; CI $[-9.64, -3.65]\,pp$). Holm adjustment across these two reported endpoints gives $p = 4.472 \times 10^{-5}$ for changed field, but the analysis remains post hoc. Reuse→full-tail has 8 exact-record Benefits and 3 Harms, a $+1.30\,pp$

change ($p = 0.2266$, CI $[-0.26, 3.13]$ *pp*). The separate controller in Appendix C.1 uses different data.

Table 10: Natural v2 formal three-arm result on 384 independent dialogues.

Metric	Dense	Reuse	Full-tail
Exact record	58 (15.10%)	54 (14.06%)	59 (15.36%)
Changed field	256 (66.67%)	231 (60.16%)	257 (66.93%)
Parseable	383	384	383
Nonblank	384	384	384
Total prefill tokens	61,440	6,144	28,048
Mean prefill tokens	160.00	16.00	73.04
P50 latency (s)	8.231	8.069	8.102
P95 latency (s)	14.861	14.561	14.639

D.1 Runtime edit and normalization conventions

The method receives only the reference and target token sequences, runtime edit range and execution plan, parsed reuse record, runtime schema keys, and parser/schema validity. It rejects gold state, gold changed-field identity, arm correctness, Benefit/Harm, and Dense or full-tail output. Minimal old and new fragments are constructed deterministically by removing the longest common token prefix and suffix from the reference and target sequences; the residual spans must fall inside the declared edit range.

The match normalizer applies the following operations in order: Unicode NFKC; Python `lower()` (not `casefold()`); `strip()`; replacement of each Unicode punctuation-category character by a space; and whitespace collapse using `split`/`join`. It uses no ontology, numeric alias table, learned parameter, or external model. This specification is distinct from scoring in Appendix A.3.

Writeback uses NFKC followed by `strip()` and `split`/`join`. It preserves case and punctuation, and copies unselected fields without modification. The verification bundle contains this writeback implementation and records the matching-normalizer order in its code-verification note.

D.2 Exact routing and direct patching

For each valid structured output, all schema field values are searched. Use the same M_o and M_u defined in Equation 1; the old and new normalized fragments must be nonempty and distinct.

- **RevisionGuard-Exact** routes to full-tail iff $|M_o| = 1$ and $|M_u| = 0$. Invalid parses, empty or colliding fragments, multiple old matches, and outputs containing both values retain reuse. The diagnostic Exact+Invalid additionally routes parser- or schema-invalid outputs.
- **RevisionPatch-Exact** uses the same syntactic match condition, replaces the complete uniquely matched field with the NFKC- and whitespace-normalized surface specified in Section 3, and otherwise returns the reuse output byte-for-byte. It never calls full-tail.

RevisionGuard-Exact was frozen before formal execution. RevisionPatch-Exact was frozen after formal GPU execution began but before any partial formal output or gold-linked correctness result was inspected. This timing distinction is intentional and is preserved in the preregistrations.

D.3 Fixed fuzzy guard and diagnostic variants

RevisionGuard-EditCoverage has no numerical threshold. Parser/schema-invalid records route to Full-tail. Valid records with empty or normalization-colliding edit fragments retain Reuse. Otherwise it routes to Full-tail when no complete field matches the normalized new fragment. It need not observe an old-value match.

Static stale exposure ranks requests by their executed stale-suffix token count in descending order, breaking ties by case ID. It selects the top $\lceil 0.25N \rceil$ requests, giving exactly 96 Full-tail actions for $N = 384$. This is an actual action-count rule, unlike the controller's calibration-budget labels.

RevisionGuard-EditMargin was frozen before v2 model outputs were inspected. For fragment text z and output value v, it computes

$$s(z, v) = \max\{\text{SequenceMatcher}(z, v), \text{Jaccard}(\text{tok}(z), \text{tok}(v))\}. \quad (5)$$

Both inputs to s are match-normalized. SequenceMatcher uses its character ratio with `autojunk=False`; tok denotes whitespace-delimited token sets for Jaccard. Let s_o and s_n be the maxima over all fields for the old and new fragments. It routes to full-tail exactly when $s_o \geq 0.88$, $s_n < 0.80$, and $s_o - s_n \geq 0.15$. Empty/colliding fragments and invalid outputs retain reuse. The constants were not changed after formal outcomes.

Table 11: Frozen zero-training formal methods. "Action" is a full-tail call for guards and a field replacement for the patch.

Method	Actions	Exact	Gain	Benefit/Harm
RevisionGuard-Exact	51	59/384	+1.30 pp	5/8, 0/3
RevisionGuard-EditCoverage	153	60/384	+1.56 pp	7/8, 1/3
RevisionGuard-EditMargin	54	61/384	+1.82 pp	7/8, 0/3
Static stale-suffix top 25%	96	55/384	+0.26 pp	3/8, 2/3
RevisionPatch-Exact	**51**	**63/384**	**+2.34 pp**	**9 fixes, 7 field harms**

EditMargin's lift over matched-count random routing is $+1.64$ pp(cluster-bootstrap CI $[0.59, 2.91]$ pp; one-sided fixed-seed randomization $p = 0.00010$). Its offline selected-arm cost is 9,386 prefill tokens, or 24.442708 per request; the live two-pass total is 10,250 (Appendix F). RevisionPatch-Exact improves exact accuracy from 54/384 to 63/384 (McNemar $p = 0.003906$; CI $[1.04, 3.91]$ pp) and changed-field accuracy from 231/384 to 269/384 without prefill. The frozen rule has nine exact fixes and zero exact breaks, but the later all-field audit finds seven wrong-field actions that introduce errors. Dense and Full-tail each obtain 4/0 exact fixes/breaks, so the nine Reuse fixes cannot all be attributed to stale KV. An exploratory fuzzy patch remains secondary.

D.4 Post-hoc prior-record and erratum comparisons

These baselines were fixed after reviewer feedback and are post-hoc comparisons, not part of the original primary protocol. The original serving path generates at most one token on the reference request to materialize KV and does not retain a parsed structured record. For the counterfactual prior-record baseline, we therefore generate the complete reference record, freeze the same exact old→new action before gold, and then score the updated record. In the recorded workflow, a full reference-record generation precedes an independent erratum target call. The latter keeps all 160 reference tokens as an exact prefix, appends a fixed correction template, uses standard prefix caching, and applies no output patch. It neither generates nor consumes a Reuse output; it is not a second target generation after Reuse.

Table 12: Prior-record and erratum comparisons on the 384 primary Qwen3 cases. Fix/break is relative to Reuse exact correctness.

Method	Parse	Exact	Changed	Fix/break	Incremental work
Reuse	384	54	231	—	baseline generation
Reuse+whole-field Patch	384	63	269	9/0	0 calls; 7 field harms
Reuse+key-aware Patch	384	63	270	9/0	trusted key; 0 field harms
Prior-record direct update	384	89	330	41/6	0 if retained
Append-only erratum	376	95	343	47/6	median +42 prompt tokens

The direct update patches 332/384 prior records; relative to Reuse+Patch it has 32/6 exact fixes/breaks. If no record is retained, its experimental full-reference generation has 8.94 s median latency and is not free. The erratum target call has 14.54 s median latency, excluding the preceding reference generation; the two medians should not be added to claim a measured end-to-end median. Its incremental prompt length has mean 42.55, median 42, and P95 48 tokens, with 95.64 mean completion tokens; relative to Reuse+Patch it has 38/6 fixes/breaks. Both have higher net accuracy than Reuse+Patch, but differ in required retained state or generation and have observed breaks. These observations do not establish

a single dominant policy. If a prior record is already available, direct updating is a relevant alternative to generating a new record and then patching it.

If the deployment already retains suitable reference KV, erratum is an alternative target-generation path using that cache. Otherwise reference-cache construction is additional work. Direct update requires the parsed prior record, which is absent from the original one-token reference-cache path. These distinctions define the comparison; no erratum latency is added to the Reuse latency.

E Patch Behavior and Long-Context Measurements

E.1 Primary actions and examples

The no-revision control uses 384 dialogue-disjoint examples without an old→new transition. RevisionPatch-Exact abstains on all of them, with zero false patch, parser failure, or exact break in the reported control. Tables 13 and 14 show the primary examples and action counts.

Table 13: Examples from the held-out evaluation. Field identities in the edit row are explanatory annotations, not patch inputs. Archived IDs: `PMUL3945.json`/10 and `MUL1706.json`/6.

Stage	Record or decision
Successful patch (MultiWOZ)	
Runtime edit	`hotel-pricerange: expensive` → `moderate`
Reuse output	`hotel-area=north; hotel-pricerange=expensive; hotel-type=guesthouse`
Why Patch fires	Valid parse; exactly one whole field equals `expensive`; no field equals `moderate`.
Patch output / gold	`hotel-area=north; hotel-pricerange=`**`moderate`**`; hotel-type=guesthouse` (wrong→correct)
Syntactic abstention (MultiWOZ)	
Runtime edit	`train-departure: cambridge` → `leicester`
Reuse output	`train-day=Sunday; train-departure=leicester; train-destination=cambridge`
Why Patch abstains	Both values are present. The revised departure is already correct, while `cambridge` legitimately remains the destination.
Gold	`train-day=sunday; train-departure=leicester; train-destination=cambridge`

Table 14: Primary action and abstention accounting. The first three rows partition all 384 natural Reuse outputs.

Decision group	Cases	Result
Unique old-only whole-field match	51	Patch; 9 exact fixes, 7 field-harm actions
Other abstention cases	324	Abstain
Old and new both present	9	Abstain
No-revision control	384	0 patch actions

E.2 Primary field-level audit and key-aware replay

The complete before/after error-field sets partition the 51 whole-field actions into nine exact fixes, 29 partial fixes, six neutral actions, and seven harmful actions. The patched key equals the intended key in 44/51 cases (86.27%); each of the seven remaining actions changes an unrelated field and introduces one error. Five of those seven cases already have a correct target field, while two leave an incorrect target unchanged. Six additional target-key actions write the runtime surface with case that differs from strict gold.

Table 15: Post-hoc policy replay on the 384 primary records. "Field harm" counts actions that introduce any new erroneous field. Key-aware assumes a trusted intended key.

Policy	Actions	Exact	Changed field	Field harm
Whole-field	51	63	269	7
Key-aware	45	63	270	0
Always abstain	0	54	231	0

Two deterministic control families reuse the 51 observed edit surfaces. When the target field is missing, schema validation makes both patch rules abstain on all 51 cases. When the target field exists but does not contain the old value and an unrelated decoy field uniquely does, whole-field matching changes the decoy in 51/51 cases and introduces an error each time; key-aware binding abstains in 51/51. These controls demonstrate a failure condition rather than estimate its natural frequency.

E.3 Post-hoc safety and coverage ablations

The safety study replays the 384 formal MultiWOZ and 384 SGD development outputs without new model calls. Each variant's actions are fixed from its observation inputs before reading gold. The analysis is post-hoc and distinct from the preregistered primary method comparison. Table 16 gives all variants from the two archived reports.

A separate all-field audit resolves the SGD aggregate endpoints action by action. The whole-field rule acts 55 times: 51 actions target the annotated field and four target another field. It yields 27 exact fixes, 21 partial fixes, and seven neutral actions. None introduces a new error because every wrongly selected target field was already incorrect; thus zero observed harm here does not imply correct field identification. Key-aware replay acts 51 times, retains the 27 exact fixes, reaches 297/384 changed fields correct, and introduces no observed field error under a supplied intended key.

Removing old uniqueness selects the first old-matching field if at least one exists and no field matches new. Removing co-occurrence abstention retains the unique-old condition but allows a new-value match. Removing schema validation searches parsed keys without the additional schema-key check. Patching every match replaces all whole-field old matches; the last-match variant changes only the last such field. Neither uses the new-value abstention condition. Global replacement instead makes case-insensitive substring replacements across field values, allowing partial-field edits. The random-field comparator uses fixed hashes to select the same number of valid-output cases as the exact rule and a field within each selected case.

Table 16: Post-hoc replay on 768 records: 384 MultiWOZ and 384 SGD development cases. F/B denotes fixes/breaks relative to each original Reuse record. Counts are descriptive sums, not an additional pooled confirmatory test. All ten reported variants and comparators are shown; Full-tail reuses the archived Full-tail outcomes.

Variant / comparator	Actions	Exact F/B	Field F/B
No repair	0	0/0	0/0
RevisionPatch-Exact	106	36/0	86/0
Remove old uniqueness	107	36/0	86/0
Remove co-occurrence abstention	129	36/0	92/0
Remove schema validation	106	36/0	86/0
Patch every whole-field match	130	36/0	93/0
Patch last whole-field match	130	36/0	93/0
Global case-insensitive substring replacement	190	38/2	104/14
Matched-count random field	106	8/8	9/2
Always Full-tail	768	27/12	60/14

The exact rule gives 36/0 exact and 86/0 annotated-field fixes/breaks. Some relaxed variants improve these endpoints, whereas global substring replacement gives 38/2 exact and 104/14 annotated-field fixes/breaks. Because this earlier replay does not compare every field's before/after error status, it

cannot establish field safety. It also does not show that each guard is individually necessary. The parser already requires a matching key set, limiting what removal of a later schema check can test.

E.4 Reuse-only transfer extensions

The disjoint post-hoc power extensions contain only Reuse generation and therefore are not complete Dense/Reuse/Full-tail cells. Mistral has 184 cases, 13 actions, and 1/0 exact fixes/breaks; Llama has 176 cases, 10 actions, and 5/0. Their respective Reuse parse failures are four and nine. We report them as Reuse-only observations rather than main-table replications.

E.5 Long-context setup and metric boundaries

The main grid contains 64 frozen natural revisions at 8k/16k tokens and concurrency 1/4. Each cell is a separate single run, with correctness and serving costs from that same run. It is not an average over the repeat diagnostic below. Each arm is a clean-start service using four Teco SDAA devices (TP4), FP16, eager execution, a 0.90 memory fraction, and 32,768 maximum batched tokens. Prefix caching is disabled; selective arms use LMCacheConnectorV1 with the one-gap overlay.

The pinned environment contains PyTorch 2.7.1, vLLM 0.17.2.dev0+g95c0f928c, TorchSDAA 3.2.0a0+gitbcfa727, LMCache 0.1.dev1082, and Transformers 4.57.3; the SDAA driver/runtime are 3.2.0/3.2.0b0. Reference caches are built before the timed target batch. Figure 3 therefore reports steady-state target performance, not startup-inclusive performance. Including reference construction gives 8,208 and 16,400 prefill tokens per Reuse target in the recorded one-reference-per-target workflow, rather than the steady-state 16. No startup-inclusive speedup is inferred from the plotted ratios.

The systems runner applies the same record handler to Dense, Reuse, and Full-tail outputs, storing correctness both before and after patching. Its final-record latency and throughput include this handler in each arm. The plotted Dense correctness is the *before-patch* count; its timing is not a separately measured handler-free Dense run. The reported ratios are retained under this explicit measurement boundary, not adjusted using the separate CPU replay estimate.

The systems adapter passes manifest old/new strings and schema keys to the patch core; it does not reconstruct a token diff on the timed path. In the supplied runner, keys are taken from the stored gold-record dictionary, while gold values are compared after the patch decision. The core receives neither gold values nor changed-field identity. This adapter tests the core given those metadata, not end-to-end runtime-only extraction of them. Their provenance is a separate reproducibility requirement.

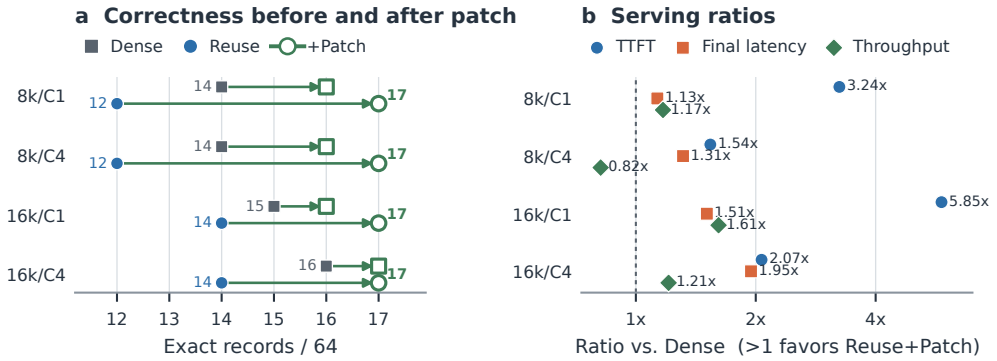

Figure 3: **Correctness and serving cost in four single-run cells.** Left: exact-record counts before and after applying the same patch to Dense and Reuse outputs; arrows show the within-arm patch increment. Right: TTFT and final-latency ratios are Dense divided by Reuse+Patch, while throughput is Reuse+Patch divided by Dense. All cells contain 64 requests. Latency includes the common record handler in both arms; reference-cache construction is excluded. C1/C4 denote concurrency.

E.6 CORRECTNESS COMPARISONS IN THE MAIN GRID

Tables 17 and 18 distinguish raw Dense, raw Reuse, Reuse+Patch, and Dense+Patch. At 16k/C1, Reuse+Patch has 17 exact records versus 15 for unpatched Dense; Patch's increment within Reuse is instead $17 - 14 = 3$. Changed-field counts are 43 versus Dense's 36, and $43 - 34 = 9$ within Reuse. These differences must not be conflated or attributed entirely to cache consistency.

Table 17: Exact-record counts on the same 64 requests in each single-run cell. RP denotes Reuse+Patch and DP denotes Dense+Patch. The two differences distinguish the patch increment within Reuse from the system-level comparison with unpatched Dense. They are not additive evidence from independent datasets.

Cell	Dense	Reuse	RP	DP	RP−Reuse	RP−Dense
8k/C1	14	12	17	16	+5	+3
8k/C4	14	12	17	16	+5	+3
16k/C1	15	14	17	16	+3	+2
16k/C4	16	14	17	17	+3	+1

Table 18: Changed-field counts on the same 64 requests in each single-run cell. RP denotes Reuse+Patch and DP denotes Dense+Patch. The two differences distinguish the patch increment within Reuse from the system-level comparison with unpatched Dense. They are not additive evidence from independent datasets.

Cell	Dense	Reuse	RP	DP	RP−Reuse	RP−Dense
8k/C1	34	30	42	39	+12	+8
8k/C4	34	30	42	39	+12	+8
16k/C1	36	34	43	39	+9	+7
16k/C4	36	34	43	39	+9	+7

Dense+Patch also improves over unpatched Dense. Its exact count is 16 in the first three cells and 17 at 16k/C4, compared with Reuse+Patch's 17 in all four. The joint main figure is therefore explicitly a system-level comparison with *unpatched* Dense, not proof that patching is useful only after reuse or that Reuse+Patch uniformly beats every patched baseline. Counts are descriptive single-run results; they are not paired significance tests.

E.7 SEPARATE CLEAN-START REPEAT DIAGNOSTICS

The repeat study uses three paired clean-start Dense/Reuse runs at each of 8k/C1, 16k/C1, 16k/C4, and 16k/C8. It does not repeat 8k/C4 and does not export correctness. For each repeat index, compute Dense/Reuse for a latency metric and Reuse/Dense for throughput; then take the arithmetic mean of the three ratios. The supplied script bootstraps those three ratio observations for mean intervals. It does not take a median of ratios or pool requests across runs before dividing. With only three runs, these intervals should be read as limited repeat diagnostics.

Table 19: Separate clean-start cost diagnostics. Each entry is the arithmetic mean of three paired-run ratios, not a median across runs or a ratio of pooled requests. Latency ratios are Dense/Reuse; throughput is Reuse/Dense. No repeated correctness result is available. There is no 8k/C4 repeat cell.

Cell	P50 TTFT	P95 TTFT	P50 final	Throughput
8k/C1	3.954	3.318	1.161	1.159
16k/C1	7.697	6.028	1.550	1.593
16k/C4	2.030	0.953	1.860	1.091
16k/C8	1.200	0.794	1.625	0.918

At 16k/C8 the mean P50 TTFT and final-latency ratios are 1.200 and 1.625, but throughput is 0.918. Mean P95 TTFT ratios are 0.953 at 16k/C4 and 0.794 at 16k/C8. These values describe means of per-run percentile ratios, not one pooled latency distribution. Repetition therefore preserves the high-concurrency limitation rather than establishing universal improvement.

E.8 COMPLETION LENGTHS

At 8k/C1, Dense versus Reuse mean/P50/P95 output tokens are 68.50/94/96 versus 67.14/94/96. At 16k/C1 they are 67.92/96/96 versus 69.00/96/96. C4 has the same P50/P95 and mean pairs 68.50/65.81 at 8k and 67.92/69.00 at 16k. These summaries document completion length but do not, by themselves, fully separate prefill and decoding cost.

F LIVE GUARD EVALUATION AND PATCH COST

Offline selected-arm cost versus live two-pass cost. Let S be the requests selected for Full-tail, c_i^R their Reuse prefill costs, and c_i^F their Full-tail costs. The offline comparison charges c_i^F for $i \in S$ and c_i^R otherwise. Live post-generation routing first runs Reuse on every request and then Full-tail on S. Therefore its total exceeds the offline total by $\sum_{i \in S} c_i^R$.

For Guard-EditMargin, $|S| = 54$ and $c_i^R = 16$. The offline total is 9,386, giving $9,386/384 = 24.442708$ tokens per request. The live total is $6,144 + 4,106 = 10,250$; the difference is $864 = 54 \times 16$. Its mean is 26.692708, not 24.44. The offline token column is not a deployable pre-decode rule: its actions are computed after observing Reuse output. It is a selected-arm accounting comparison only.

The live r99 experiment evaluated the frozen EditMargin policy and a frozen matched-count random policy in four TP2 shards each. Both made exactly 54 conditional full-tail calls. Their realized prefill totals differ slightly (10,250 versus 10,266 tokens) because the selected cases have different tail lengths; the baseline is not an exact token-budget match. For flagged cases, request-arrival-to-final-record latency includes the first reuse generation, guard decision, and second full-tail generation. Table 20 therefore does not confuse guard overhead with end-to-end latency.

Table 20: Live two-pass routing with 54/384 conditional Full-tail requests in each policy. These guard actions are distinct from an unintended runtime fallback.

Metric	EditMargin	Random	Difference
Exact record	61/384	53/384	+2.08 pp
Changed field	65.63%	60.94%	+4.69 pp
Mean latency (s)	10.55	10.45	+0.10
P50 latency (s)	9.11	9.37	−0.26
P90 latency (s)	14.77	14.64	+0.13
P95 latency (s)	29.26	19.48	+9.78
Prefill tokens	10,250	10,266	−16

The paired exact comparison has eight improvements and no regressions ($p = 0.0078125$). Mean guard time is 0.53 ms. EditMargin uses 63.5% fewer prefill tokens than always full-tail (10,250 versus 28,048), but conditional two-pass requests yield a worse P95; we therefore claim an accuracy–recomputation point, not wall-clock dominance.

Run r104 replayed every frozen RevisionPatch-Exact decision 200 times in the remote CPU environment. Decision mean/P50/P95 is 0.033/0.032/0.049 ms. Adding this measured decision cost to the recorded reuse trace gives estimated P50/P95 8.069/14.561 s with zero additional prefill. This is a trace-replay overhead estimate, not a second GPU-generation experiment.

G REVISION-HANDLING APPROACHES

Table 21: Revision-handling approaches and their required state or computation. Extra generation denotes generation beyond an already produced target response; these are deployment categories, not a head-to-head accuracy ranking.

Approach	Locus	Suffix reuse	Extra gen.	Requirement	Main limitation
Dense / full-tail recompute	Prompt / KV	No	No	None	High prefill
Selective / compositional reuse	KV plan	Yes	No	Reuse plan	Stale suffix possible
Learned KV editing	KV state	Yes	No	Training / model access	Model-specific
Verification + regeneration	Decode / output	Partial	Yes	Verifier	Added decoding
Retained-record update	App state	n/a	No	Trusted prior record	Needs retained state
Post-generation patch	**Execution / output**	**Yes**	**No**	**Runtime edit + schema**	**Unique whole-field match**