Mathematical Modeling and Algorithm Application
ISSN: 3006-0842 | Vol. 10, No. 1, 2026

Validating Repair Claims after Evidence Restoration in Compressed Prompts

Ziye Yue, Yuejin Zhang*

Smart City College, Beijing Union University, Beijing, China
* **Corresponding author**: Yuejin Zhang

Abstract: An answer that recovers after deleted evidence is restored does not, by itself, establish a reliable repair. The recovery may depend on sampling variation or on other changes to the prompt. We evaluate repair claims by fixing the restoration candidate and testing it against three distinct length-matched sham texts in fresh paired calls. An entry screen restricts search to questions that pass with the full context and fail after compression. With controlled deletion at a retention ratio of 0.5, protocol pass rates rose from 20.0% to 45.0% on 60 development items. On 100 held-out test items, they rose from 17.0% to 36.0%. Of 36 search candidates, 34 passed confirmation; one failed and one lacked enough matched texts to complete testing. All confirmed repairs were on GSM8K, with none on HotpotQA. During confirmation, target evidence was prepended and shams were appended, limiting the inference to a contrast involving both content and position. Search and confirmation together cost an estimated US$3.927; confirmation contributed 37.2%. The resulting record distinguishes candidates found during search from repairs supported by fresh control comparisons and makes the cost of that support explicit.

Keywords: Prompt Compression; Evidence Restoration; Repair Claim Validation; Matched Shams; Verification Cost

1. Introduction

Prompt compression removes input tokens to reduce the amount of text processed by a large language model (LLM) [1-3]. A question can become unanswerable if compression deletes a necessary premise. Restoring the deleted material offers a way to investigate that failure: return the evidence to the compressed context and test the answer again.

Hu et al. documented cases in which compression preserved an answer-bearing fragment but removed the referential support needed to interpret it [4]. Returning that support improved answering performance. Pair-ID examines evidence interventions with the query and reader held fixed, using shams matched for length and position [5]. These studies establish the value of restoration experiments. Whether a particular successful response warrants a repair claim still depends on the controls and the calls used to assess it.

The difficulty arises when search and verification rely on the same successful response. Repeated generation can change the answer, and adding unrelated text may also produce success. If a sham restores the answer, the target evidence has weaker support as the explanation for recovery. We therefore assess a selected restoration set against matched controls after search has ended.

Our protocol begins by identifying failures associated with compression, then searches evidence combinations constrained by their dependencies. Once a candidate is fixed, fresh paired calls compare its responses with those obtained from matched shams. We report item outcomes, baseline claim sets, and the cost of obtaining confirmation. A separate minimality record indicates whether the evidence set has also been tested for redundant members.

2. Related work

2.1. Missing evidence under prompt compression

LLMLingua compresses prompts through budget control and token selection [1]. LongLLMLingua adds query information to guide retention [2], while LLMLingua-2 learns a token classifier through data distillation for task-agnostic compression [3]. Aggregate task scores show how well these methods retain performance. For a failed question, however, the diagnostic problem is to determine which omissions changed the answer.

Referential dangling provides one account of such failures: a retained fragment can require a premise that has been deleted [4]. Testing each unit alone can overlook that dependence. We search combinations formed from dependency constraints, then use response tests to assess whether restoring a combination recovers the answer.

2.2. Counterfactual interventions and confirmation

Pair-ID studies the same failure by adding supporting evidence and removing verified nonsupporting material, with matched shams auditing the responses [5]. Prompt Adaptation using Interventional Rollouts (PAIR) instead fixes the agent state and examines compression events that affect later execution [6]. In each case, the intervention can be interpreted only with reference to the conditions held fixed.

Joint Prompt Attribution (JoPA) examines prompt fragments whose joint presence influences generation [7]. Such interactions motivate retaining evidence combinations as search objects. They also leave a selection problem: calls that identify a promising combination have already influenced which candidate is reported. We use those calls for exploration and assess the fixed candidate with fresh calls on the same question.

2.3. Criteria for restoration sets

The ddmin delta debugging algorithm reduces an input to a locally minimal failure-inducing set [8]. Probabilistic Monotonicity Assessment (PMA) estimates monotonic relationships from earlier tests to reduce search cost [9]. Restoration need not behave monotonically: an added unit may complete a premise or interfere with it. Our search therefore caches exact tested combinations; a failed combination does not rule out its untested extensions.

Confirmation asks whether the restoration arm succeeds reliably and whether it outperforms the sham arm. We use Wilson bounds for the success thresholds [10] and a paired bootstrap for the difference between arms [11]. An exact McNemar test describes the binary change in item outcomes after confirmed restoration [12]. These analyses answer different questions: candidate tests assign repair labels, whereas item tests quantify the change in protocol pass rates.

3. Repair claim validation protocol

3.1. Screening for compression-related failures

Let C denote the full context, C' the compressed context, and Q the question. Deleted material is partitioned into evidence units,

$$U = \{u_1, \dots, u_m\}. \tag{1}$$

A restoration candidate R is a subset of this collection, inserted into the compressed context through a fixed template. The judge J maps each model answer to match, mismatch, or abstention.

We sample the full and compressed contexts 15 times each to obtain success rates $\hat{p}_{\text{full}}$ and $\hat{p}_{\text{comp}}$, using an entry threshold of 0.8. The full-context Wilson interval is checked first. If it crosses the threshold, the item is labelled unstable in full-context answerability. Of the remaining items, those below threshold in both conditions exit attribution, and those passing with compressed context receive the status no failure observed. Restoration search accepts items that meet the full-context threshold but fall below it after compression.

This screen establishes the baseline against which restoration is tested. The full-context stability check uses the implementation's fixed normal quantile of 1.6449.

3.2. Generating restoration candidates

A candidate must include the evidence units needed to interpret its members. Starting with a seed set, dependency closure adds the other members of each jointly necessary edge and the prerequisites of each conditional edge. Alternative sufficient branches remain separate, so closure preserves different possible restoration paths.

Search permits at most three evidence units per candidate. A closure beyond this limit is recorded as over-capacity. Among executable candidates, smaller unit counts take priority, followed by shorter length. A cache of exact tested combinations prevents the same combination from being tested again.

3.3. Comparing restoration with matched shams

Exploration appends either candidate R or sham text S to the compressed context. The confirmation script uses a different placement: target evidence precedes the compressed context, while the sham follows it. Both arms use the same compressed context, question, answer instruction, and generation parameters. Texts are matched according to

$$|\tilde{t}(S) - \tilde{t}(R)| \le \tau \tilde{t}(R). \tag{2}$$

The length proxy $\tilde{t}$ is computed offline, with tolerance τ set to 5%. For English text, character count is divided by four and rounded up, with a minimum value of one. Proxy lengths are summed for candidates containing multiple evidence units. Matching uses this approximation; token costs use the provider's returned usage.

We select shams from a frozen pool of natural English texts. Before ranking by length difference, we exclude texts sharing numerical values with the question or compressed context. A candidate requires three distinct shams, with 20 call pairs assigned to each. The same pool text may serve different candidates. We record both coverage per candidate and the number of unique texts used in each split.

This matching rule constrains added length under the proxy, but confirmation changes insertion position together with content. Its treatment-control contrast therefore concerns the two executed input constructions. A success shared by both arms offers less support for evidence specificity.

3.4. Confirming fixed candidates with fresh calls

For each fixed candidate, confirmation runs 60 call pairs, divided into 20 pairs per sham text. The restoration set and insertion procedure remain fixed throughout. A fixed seed determines pair order and which arm runs first. Here, independent confirmation means that the calls are separate from search; the question is unchanged.

The treatment success rate must satisfy

$$L(\hat{p}_{\text{treat}}) \ge \theta_{\text{conf}}. \tag{3}$$

The confirmation threshold θ_{conf} is 0.7. The sham arm must satisfy

$$U(\hat{p}_{\text{sham}}) < \theta_{\text{conf}}. \tag{4}$$

The paired difference must also retain a positive lower bound,

$$L_{\text{boot}}(\hat{p}_{\text{treat}} - \hat{p}_{\text{sham}}) > 0. \tag{5}$$

We compute one-sided 95% Wilson bounds for equations (3) and (4). For equation (5), we resample call pairs 2,000 times with seed 2026 and take the fifth percentile as the one-sided lower bound. All three criteria must hold for a confirmed-repair label. Insufficient matched texts leave a candidate pending; a completed test that misses a criterion is labelled failed confirmation.

Repeated calls measure response variation for a fixed input, and the three sham texts broaden the control comparison. Candidate confidence bounds and binomial tests assume exchangeability of repeated calls. When estimating item-level uncertainty, we resample questions and keep each question's repeated calls attached to that item.

3.5. Separating effectiveness from local minimality

Effectiveness confirmation tests whether restoration supports stable answering. Local minimality asks whether any member of the restoration set is redundant. During search, each member e is removed in turn,

$$R_{-e} = R \setminus \{e\}. \tag{6}$$

A locally irreducible flag requires every tested member deletion to fall below the development threshold. For a single-unit candidate, the entry result without restoration supplies the reference. A two-unit candidate needs member-deletion retests. This attribute is recorded independently of the confirmed-repair label, which concerns answering

effectiveness.

3.6. Budget limits and recorded states

Before testing a search candidate, we check the remaining allowances for calls, tokens, cost, and runtime. A shortage stops the test before execution. Search is capped at 45 calls per item. Fresh confirmation checks cumulative calls and cost after each returned pair and keeps its own account of completed pairs. Table 1 lists search states and their handling.

Table 1 Search states and subsequent handling

Search state	Interpretation	Subsequent handling
No failure observed	Compressed context meets the entry threshold	Retain the original answer
Compression-related failure	Entry criteria met; intervention incomplete	Search for candidates
Search-confirmed repair	Validation conditions within the search pipeline are complete	Retain validation records
Repair awaiting confirmation	Candidate passes exploration	Confirm with fresh calls
Partial validation	Recovery observed; some attributes unresolved	Retain attributes and confirm effectiveness
Budget exhausted	A budget limit is reached	Stop and retain results
Indeterminate	Entry instability or insufficient search evidence	Exit with a reason code

The internal search state search-confirmed repair is distinct from a label obtained through fresh-call confirmation. Some historical runs labelled pretest budget shortages as indeterminate. We retain those original states in aggregation and identify budget stops through reason codes. The sequence and separate minimality branch are shown in Figure 1.

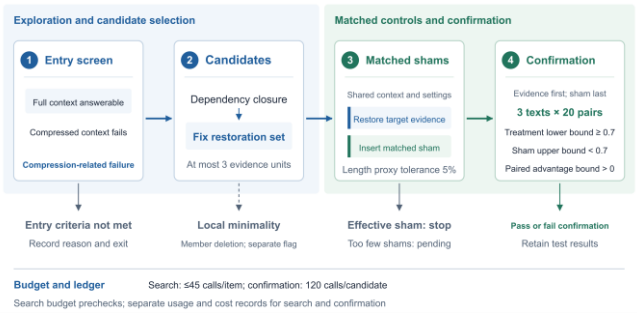


Figure 1 Repair claim validation protocol

Note: Fresh confirmation prepends target evidence and appends shams. Exploration appends either arm to the compressed context. The dashed branch denotes the separately recorded minimality attribute.

4. Experimental design

4.1. Data and execution conditions

Development uses 30 HotpotQA [13] and 30 GSM8K [14] items; held-out testing uses 50 from each dataset, for 160 items overall. We group items by source paragraph and keep each group within a single split. Parameters were fixed before the test split was unsealed. The subsequent confirmation runs used the same rules.

We apply controlled deletion at a retention ratio of 0.5. The provider identifies the answering model as deepseek-flash; generation uses temperature 0.3 and a maximum of 512 output tokens. A common programmatic judge scores every condition. It extracts the final number and applies a tolerance comparison for numerical answers, checks normalized containment for entities, and takes the first yes/no response for Boolean answers.

4.2. Baselines and outcome definitions

We compare six restoration baselines. Random reinsertion selects without ranking; Best Matching 25 (BM25) reinsertion and node ranking prioritize candidates. The remaining procedures are ddmin restoration, monotonic pruning, and dependency-aware ddmin. Each baseline claims a repair when at least four of five sampled answers match. Our confirmed set instead follows entry screening and matched confirmation. We align claims by item when comparing the resulting sets.

For item i, let $C_{0,i}$ indicate the compressed baseline status no failure observed, and let A_i indicate passing three-text confirmation. Both are binary. The post-restoration indicator is

$$C_{1,i} = \max(C_{0,i}, A_i). \tag{7}$$

Equation (7) retains baseline passes and adds confirmed repairs. The corresponding means are termed protocol pass rates. With N items, the initial failure count is

$$N_{\text{fail}} = N - \sum_i C_{0,i}. \tag{8}$$

The confirmed repair rate uses this denominator,

$$\rho = \frac{\sum_i A_i}{N_{\text{fail}}}. \tag{9}$$

The paired improvement is

$$\Delta = \frac{1}{N}\sum_i (C_{1,i} - C_{0,i}). \tag{10}$$

Equation (7) fixes the regression count at zero by construction. Dataset-specific results identify which items contribute to the improvement in equation (10).

4.3. Statistical analysis and cost accounting

Two-sided 95% Wilson confidence intervals (CIs) accompany item-level proportions. For paired improvements, we use percentile intervals from 10,000 item-level bootstrap resamples, with seed 147258. Development analyses are exploratory. Held-out test outcomes are evaluated under the rules fixed before that split was opened.

Candidate-level one-sided binomial tests use the null hypothesis

$$H_0: p \le 0.7. \tag{11}$$

In this test, p denotes success probability for the fixed candidate's treatment condition. We also perform a post hoc Bonferroni analysis, treating the 15 development candidates and 21 test candidates as separate families.

We price usage at frozen rates of US$0.27 per million input tokens and US$1.10 per million output tokens, separating search from confirmation. Individual call or call-pair records supply usage except for 312 historical calls on 15 test items. Only item-level totals survive for those calls. We estimate

their costs from the same split's per-token cost and identify them separately in the accounts.

5. Results

5.1. Sham controls identified synthetic confounding

Of 25 deterministic synthetic worlds, 18 yielded confirmed repairs and seven exited the procedure. Two worlds were confounded: unrelated text also caused success. All restoration baselines claimed repairs in these worlds, but sham successes caused the protocol to exit. These cases check that the confounding branch operates as intended. They are mechanism checks and are kept separate from the model experiments.

5.2. Confirmed repairs increased protocol pass rates

Fresh confirmation added 15 passes in development and 19 in held-out testing. Pass counts increased from 12 to 27 and from 17 to 36, respectively. Table 2 gives the corresponding pass rates, paired improvements, and repair rates among initial failures.

Table 2 Item-level protocol pass rates and confirmed repair rates

Split	Items	Compressed baseline	After restoration	Improvement [95% CI]	Confirmed repair rate
Development	60	20.0%	45.0%	25.0 [15.0, 36.7]	31.3%
Test	100	17.0%	36.0%	19.0 [12.0, 27.0]	22.9%
Combined	160	18.1%	39.4%	21.3 [15.0, 27.5]	26.0%

Paired improvements are expressed in percentage points. Among the 48 initial development failures and 83 test failures, the confirmed repair rate CIs were 20.0%-45.3% and 15.2%-33.0%. Two-sided exact McNemar p values were 0.0000610 for development and 0.00000381 for testing. Combining the splits gave 0.000000000116.

GSM8K supplied every confirmed repair. Its pass rate increased from 10.0% to 60.0% in development and from 10.0% to 48.0% in testing. HotpotQA stayed at 30.0% and 24.0%, respectively (Figure 2), with no confirmed repair. The pooled gain thus describes recovery on numerical reasoning items in this experimental setting.

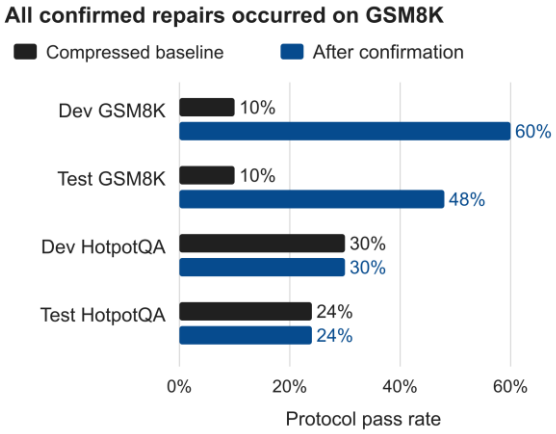


Figure 2 Protocol pass rates by dataset and split

Note: Each dataset contributes 30 development items and 50 test items. Bars show descriptive item-level pass rates; Dev denotes development. Table 2 reports intervals for item-level improvements.

There were 33 indeterminate development items and 62 indeterminate test items. Failure with both full and compressed context accounted for 14 and 27, respectively; budget stops accounted for 12 and 27. Full-context instability accounted for six development and eight test items. One further development item exhausted the sham pool.

5.3. Fresh calls confirmed 34 of 36 search candidates

Search returned 36 candidates. Of these, 35 had three eligible sham texts and completed confirmation. Treatment successes totalled 896 in development and 1,166 in testing, with no sham successes in either split (Table 3). Reuse across candidates reduced the number of unique sham texts to 28 in development and 31 in testing.

Table 3 Candidate outcomes and responses during three-text confirmation

Split	Search candidates	Completed	Passed	Call pairs	Treatment successes	Sham successes
Development	15	15	15	900	896	0
Test	21	20	19	1,200	1,166	0
Total	36	35	34	2,100	2,062	0

The failed test candidate contained two units and succeeded in 35 of 60 treatment calls. A different candidate had only one eligible short sham and could not complete confirmation. Accordingly, 97.1% of completed candidates passed, compared with 94.4% of all search candidates. Figure 3 plots the response rates of completed candidates.

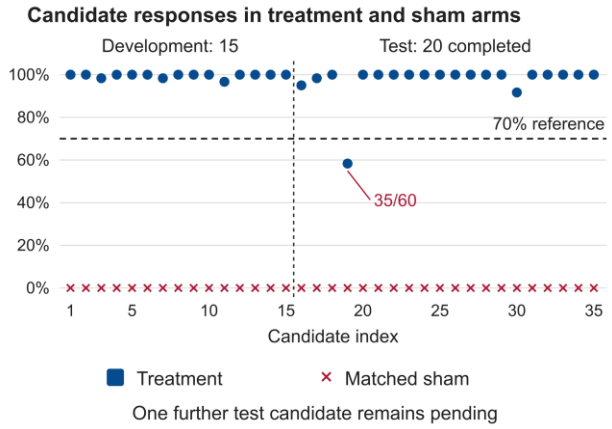


Figure 3 Candidate-level response rates during confirmation

Note: Each point represents one completed candidate, with 60 calls in each arm. The dashed 70% line is a raw-rate reference. Passing also requires the confidence-bound and paired-advantage criteria in equations (3)-(5). The candidate with insufficient matched texts is excluded from the horizontal axis.

Bonferroni correction left all 34 passing candidates significant in the treatment-arm binomial tests; the largest adjusted p value was 0.001051. The confirmed sets contained 27 single-unit and seven two-unit candidates. The latter seven have evidence for effectiveness but remain pending for local minimality. Member-deletion retests stopped at the budget precheck for all nine two-unit search candidates. We therefore retain minimality as a separate, unresolved attribute.

5.4. Baselines produced more claims than the confirmed set

The baseline rule, based on five samples, identified 44 common development failures; our 15-sample entry screen identified 48 initial failures. Four items differed near the full-context stability or compressed-pass thresholds. To compare claims on the same items, we restricted baseline sets to the common 44 failures and aligned them with our 15 confirmed repairs.

Table 4 Composition of baseline claims on 44 common development failures

Method	Claims	Overlap with confirmed set	Additional claims	Additional share	Runtime errors
Random reinsertion	30	14	16	53.3%	0
BM25 reinsertion	28	14	14	50.0%	0
Node ranking	28	15	13	46.4%	0
ddmin restoration	27	15	12	44.4%	0
Monotonic pruning	24	14	10	41.7%	4
Dependency-aware ddmin	29	15	14	48.3%	0

Across baselines, claim counts ranged from 24-30, or 1.6-2.0 times the protocol's confirmed count. Of these, 14-15 claims overlapped with confirmed items. The remaining 10-16 represented 41.7%-53.3% of each baseline's claims (Table 4; Figure 4).

Item overlap compares the outputs of the decision rules; it does not confirm the particular evidence set selected by a baseline. Claims outside our confirmed set likewise need their own confirmation before they can be accepted or rejected. We count the four monotonic-pruning memory errors separately and calculate additional shares using formed claims as the denominator.

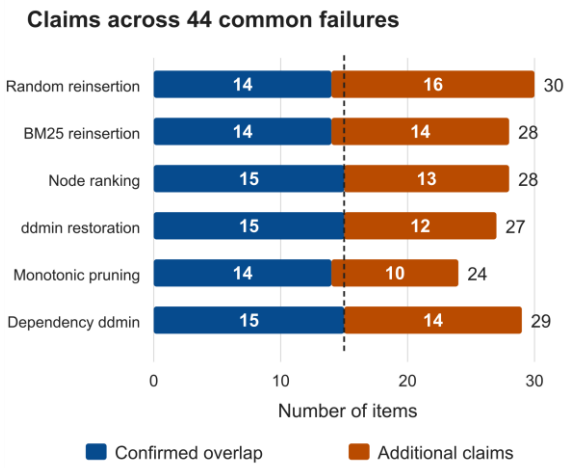


Figure 4 Item overlap between baseline claims and the confirmed set

Note: The comparison covers 44 common development failures. The vertical dashed line marks the count of 15 confirmed repairs. Overlap is defined by item; additional claims fall outside this study's confirmed item set.

5.5. Repeated search retained the same candidate sets

We reran a stratified subset of 30 items with seeds 147258, 147259, and 147260. Twenty-nine completed every run. For 26 of them, final search states agreed across seeds, giving 89.7% agreement. Eight items returned identical restoration sets in all three runs, while the other 21 returned no candidate. Three HotpotQA items switched between no failure observed and indeterminate, placing the observed state variation at the entry boundary.

The search logs show a separate role for sham tests. An effective sham blocked one development candidate, and three further candidates across the splits lacked enough control texts. Search-stage sham testing consumed 285 calls at US$0.159, or 6.4% of search cost.

5.6. Confirmation accounted for over one third of total cost

Search required 5,289 calls across the two splits; confirmation required another 4,200. Table 5 separates their recorded token usage and estimated costs under the frozen prices.

Table 5 Recorded usage and estimated costs for search and confirmation

Split and stage	Calls	Tokens	Estimated cost US$
Development search	2,097	1,844,759	0.933
Development confirmation	1,800	699,012	0.600
Test search	3,192	3,016,685	1.533
Test confirmation	2,400	1,000,313	0.860
Total	9,489	6,560,769	3.927

Displayed costs are rounded to three decimal places; totals are calculated before rounding. The test-search account includes 312 historical calls, covering 328,656 tokens on 15 items, at an estimated cost of approximately US$0.167.

Entry screening cost US$0.807 in development and US$1.228 in testing, corresponding to 86.5% and 80.1% of recorded search cost. Figure 5 identifies historical test usage separately because its cost was reconstructed from item-level totals.

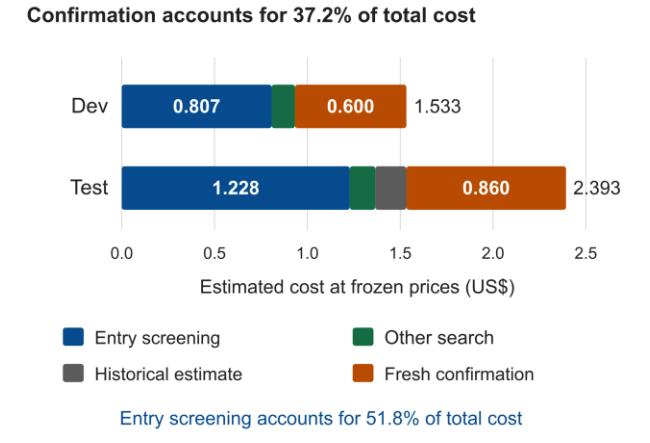


Figure 5 Cost composition across search and confirmation

Note: Bars stack estimated costs by stage; Dev denotes development. Gray denotes historical usage estimates. Cost shares use unrounded totals for the complete procedure.

Across search and confirmation together, entry screening contributed 51.8% of cost and confirmation contributed 37.2%. Development cost US$1.533 and testing US$2.393, equivalent to US$0.102 and US$0.126 per confirmed repair. Each confirmation used 120 calls, approximately 3.4-3.8 times the mean search calls per item. Screening incurs expense on every item, whereas confirmation devotes repeated calls to the selected candidates. Both expenses enter the cost of a verified claim.

6. Discussion

A successful search response is useful for finding a candidate, but it is only the starting point for verification. In this protocol, the restoration set is fixed before fresh calls test its advantage over shams. Entry screening first defines the failure to be investigated. Keeping these decisions separate makes it possible to distinguish a promising restoration from one that passes the stated confirmation criteria.

Evidence restoration and matched intervention audits already provide tools for diagnosing missing support [4,5]. Our addition is to make control responses part of the decision to confirm a candidate. The larger baseline claim sets illustrate the consequence of choosing a different decision rule. Agreement on an item cannot transfer confirmation to another restoration set. Candidate counts should therefore be reported with their confirmation outcomes, so that the evidence behind each reported repair remains visible.

The scope of the result is narrow in two respects. Every confirmed repair came from GSM8K, so the gains cannot be extended to HotpotQA from these experiments. In addition, confirmation prepended evidence but appended shams, changing content and position together. The 34 passing records establish performance under that contrast and cannot isolate the contribution of evidence content. The judge and length proxy also delimit what was measured. For the candidates awaiting minimality retests, effectiveness does not establish the absence of redundant evidence.

Verification also changes the resource requirements of a restoration experiment. Screening dominates search expenditure because every item must first be assessed, including items that never produce a candidate. Confirmation then spends additional calls on the selected sets. Separate cost records distinguish the expense of establishing a failure from that of checking a selected restoration. Repair counts and verification costs should be considered together when judging the practical scope of compression diagnostics.

7. Conclusion

We distinguish evidence-restoration candidates from repairs confirmed with fresh paired calls and matched shams. Under the three-text rule, 34 of 36 candidates passed confirmation, one failed, and one remained pending. Each outcome is recorded separately from the state reached during search.

With controlled deletion at retention ratio 0.5, protocol pass rates increased by 25.0 percentage points in development and 19.0 in held-out testing. These gains came entirely from GSM8K and remain subject to the combined content-position contrast. The estimated procedure cost was US$3.927. Reporting the fixed candidate, its control responses, and the

cost of confirmation gives a reproducible account of repair claims under the tested conditions.

References

[1] Jiang, H., Wu, Q., Lin, C. Y., et al. (2023). LLMLingua: Compressing prompts for accelerated inference of large language models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 13358–13376. https://doi.org/10.18653/v1/2023.emnlp-main.825

[2] Jiang, H., Wu, Q., Luo, X., et al. (2024). LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1658–1677. https://doi.org/10.18653/v1/2024.acl-long.91

[3] Pan, Z., Wu, Q., Jiang, H., et al. (2024). LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. Findings of the Association for Computational Linguistics: ACL 2024, 963–981. https://doi.org/10.18653/v1/2024.findings-acl.57

[4] Hu, Z., Li, K., Fu, D., et al. (2026). Relevant but incomplete: Referential dangling as a paradigm-level failure mode in hard prompt compression. arXiv preprint arXiv:2608.04569. https://doi.org/10.48550/arXiv.2608.04569

[5] Du, W. (2026). What would fix this RAG failure? Auditing counterfactual response with paired evidence interventions. arXiv preprint arXiv:2608.08944. https://doi.org/10.48550/arXiv.2608.08944

[6] Min, G., Wu, L., Shi, M., et al. (2026). Adapting context compression for long-horizon agents with counterfactual continuations. arXiv preprint arXiv:2609.36526. https://doi.org/10.48550/arXiv.2609.36526

[7] Chang, Y., Cao, B., Wang, Y., et al. (2025). JoPA: Explaining large language model's generation via joint prompt attribution. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 22106–22122. https://doi.org/10.18653/v1/2025.acl-long.1074

[8] Zeller, A., & Hildebrandt, R. (2002). Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering, 28(2), 183–200. https://doi.org/10.1109/32.988498

[9] Tao, Y., & Xue, J. (2025). Accelerating delta debugging through probabilistic monotonicity assessment. Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering, 12–22. https://doi.org/10.1145/3756681.3756940

[10] Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. Journal of the American Statistical Association, 22(158), 209–212. https://doi.org/10.1080/01621459.1927.10502953

[11] Efron, B., & Tibshirani, R. J. (1993). An introduction to the bootstrap. Chapman & Hall. https://doi.org/10.1007/978-1-4899-4541-9

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

[13] Yang, Z., Qi, P., Zhang, S., et al. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2369–2380. https://doi.org/10.18653/v1/D18-1259

[14] Cobbe, K., Kosaraju, V., Bavarian, M., et al. (2021). Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168. https://doi.org/10.48550/arXiv.2110.14168