GLITCHPATCH: REPAIRING GLITCH TOKENS IN FROZEN LANGUAGE MODELS VIA LOCAL RETOKENIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Glitch tokens are anomalous vocabulary entries that can cause large language models (LLMs) to produce outputs inconsistent with their inputs. Existing repair methods require access to model internals, making them impractical for frozen checkpoints. We investigate whether glitch tokens can be repaired outside the model by optimizing the input tokenization. An empirical study on BPE merge-rule deletion reveals that (1) deleting a glitch token's merge rule can fix a substantial fraction of failures, yet disrupting normal tokens sharing intermediate merge nodes causes the overall glitch rate to rise, and (2) different decomposition granularities yield non-monotonic fix rates while collateral damage on normal tokens grows monotonically. Motivated by these findings, we propose **GLITCHPATCH**, a repair framework for frozen language models based on local retokenization, consisting of two stages: the **offline stage** uses Behavioral Path Optimization (BPO) to find the behaviorally optimal replacement token sequence for each glitch token and compiles validated replacements into a rule table; the **online stage** substitutes only the IDs of matched glitch tokens in the canonical token sequence, with no modification to model parameters or internal states. Experiments on ten models spanning six tokenizer families show that GLITCHPATCH achieves an 85.10% mean fix rate, outperforming the strongest baseline by 14.37 percentage points, and reduces the average glitch rate from 14.88% to 2.27%. GLITCHPATCH achieves a 0.00% RR in full-vocabulary evaluation and leaves rule-unmatched inputs unchanged by design. We further evaluate the practical impact of repair from the perspectives of time cost, language understanding, and capability, supporting its deployment feasibility. We hope this work provides a practical option for improving tokenizer reliability.

1 INTRODUCTION

The vocabularies of large language models (LLMs) contain anomalous entries known as *glitch tokens*, which can induce outputs inconsistent with their inputs (Land & Bartolo, 2024). Some glitch tokens closely resemble normal tokens, making them easy to trigger inadvertently and leading to unexpected behavior that degrades user experience (Figure 1). When such strings appear in product names, database fields, or code identifiers, errors can propagate to downstream tasks. Prior work has linked glitch tokens to unsafe generation and degraded instruction following (Land & Bartolo, 2024; Li et al., 2024; Tang et al., 2026), making their repair a necessary step toward improving model reliability.

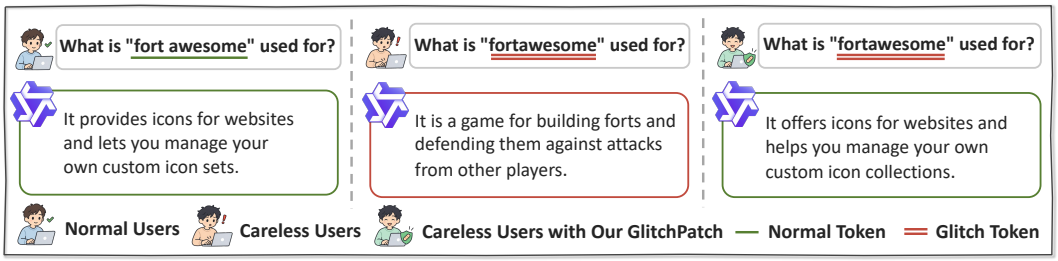

Figure 1. A glitch-token example and our GLITCHPATCH repair. The strings "fortawesome" (glitch token) and "fort awesome" (normal tokens) differ by one space. The model describes an icon library for the latter but a fort-building game for the former. GLITCHPATCH restores the relevant answer without changing the input text.

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.

Existing work repairs glitch tokens from two directions. Activation-based methods, represented by GlitchProber (Zhang et al., 2024), access intermediate-layer activations at inference time and correct target neuron states. Parameter-based methods modify the model's parameters: GlitchEdit (Tang et al., 2026) edits the embedding weights of target tokens, and GlitchCleaner (Fan et al., 2026) trains and attaches a gated LoRA branch. These methods are effective in their respective settings, but they must happen **inside** the model. Whether inserting activation hooks, rewriting embedding weights, or loading adapter parameters, all require access to model internals or alter the model's parameters. In practice, however, most providers deploy frozen checkpoints integrated into production pipelines with verified kernels, version control, and regression testing. Any internal modification can invalidate these pipelines and introduce regression risks. When providers discover glitch tokens, they typically need a fast, reversible fix that leaves the model itself untouched rather than a costly internal refactor. **To date, no work has explored whether glitch tokens can be repaired from outside the model.**

Tokenization provides a natural entry point for external repair. An LLM does not process raw text directly; its tokenizer encodes the input into a sequence of vocabulary tokens, each mapped to a learned embedding (Figure 2). Tokenizers typically use byte-pair encoding (BPE), which builds tokens by progressively merging shorter units according to learned merge rules. If the final token is a glitch token, an intuitive fix is to block its last merge so that the model receives the normal sub-tokens instead. The observation that frozen models can tolerate non-canonical tokenizations supports this direction (Zheng et al., 2025). However, this intuition does not directly yield reliable repair. We conduct an empirical study on deleting BPE glitch token merge rules and uncover two key findings (Figure 3). First, deleting a glitch token's merge rule does fix a portion of failures but also disrupts normal longer tokens that share the same intermediate merge node (Figure 4a), causing the overall glitch rate to increase (**Finding 1**). Second, decomposing a glitch token to different granularities produces replacement paths whose fix rates vary non-monotonically across models (Figure 4b), while collateral damage to normal tokens grows monotonically with depth (**Finding 2**). These two findings expose two open problems for external repair: **how to find the optimal replacement path for each glitch token, and how to eliminate collateral effects on normal tokens.**

Motivated by these findings, we propose **GLITCHPATCH**, an external repair framework for frozen language models based on Local Retokenization (Figure 5). GLITCHPATCH leaves the tokenizer unchanged and replaces only the IDs of matched glitch tokens in the canonical token sequence. The framework has two stages: **offline**, Behavioral Path Optimization (BPO) searches for the behaviorally optimal equivalent token sequence for each known glitch token and compiles validated replacements into a rule table through full-context evaluation and independent calibration; **online**, only rule matching and ID substitution are performed, without reading or modifying any model parameters or internal states. Experiments on ten models spanning six tokenizer families show that: ❶ for repair effectiveness, GLITCHPATCH achieves an 85.10% mean fix rate, outperforming the strongest baseline by 14.37 percentage points and reducing the mean glitch rate from 14.88% to 2.27%; ❷ for impact on normal tokens, the regression rate is 0.00% on the full-vocabulary evaluation, while requests that trigger no applicable rule are preserved by construction; ❸ for practicality, the average token-level loss drops by 13.5% on natural texts containing glitch tokens, with negligible online latency overhead. These results demonstrate that by optimizing the tokenization, an external method can not only repair glitch tokens effectively but outperform existing internal approaches.

In summary, this paper makes three contributions: ❶ we present the first exploration of repairing glitch tokens from outside the model, identifying two limitations of merge-rule deletion that motivate our approach (Section 3); ❷ we propose GLITCHPATCH, a framework that combines offline BPO path optimization with online local retokenization, repairing glitch tokens with minimal impact on normal tokens and overhead (Section 4); ❸ extensive experiments show that GLITCHPATCH achieves a mean fix rate of 85.10%, surpassing all existing methods, with validated practical deployment feasibility in time cost, language understanding, and general capability (Section 5 & 6).

2 BACKGROUND AND RELATED WORK

This section introduces tokenization in LLMs and reviews glitch-token detection and repair, clarifying the gap between existing methods and frozen deployment needs.

2.1 TOKENIZER AND TOKENIZATION

LLMs do not process raw text directly. The input is first split by a tokenizer into vocabulary tokens, each mapped to a vector via an embedding lookup and then processed by the Transformer. A token may be a word fragment or part of a byte sequence, so different segmentations of the same text produce different model inputs.

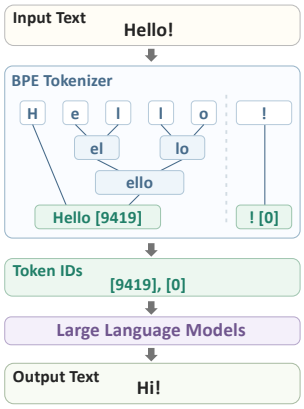

Figure 2. BPE tokenization of "Hello!" with Qwen3.8-27B.

Most modern tokenizers adopt Byte Pair Encoding (BPE). BPE learns an ordered set of merge rules from the training corpus; at encoding time, it starts from the smallest units and repeatedly merges adjacent pairs by rule priority until no further merge applies (Figure 2, Qwen3.8-27B(Qwen Team, 2026)). For a given text and tokenizer, this deterministic result is the canonical tokenization, denoted $T(x)$. However, blocking a merge step or using other vocabulary entries covering the same byte span can produce ID sequences that decode to the same bytes; we call these equivalent tokenizations. Prior work has shown that some frozen models tolerate or even benefit from non-canonical segmentations (Zheng et al., 2025; Ghosh & Jyothi, 2026), and techniques such as BPE-Dropout enhance this robustness by introducing segmentation variation during training (Kudo, 2018; Provilkov et al., 2020). These findings indicate that tokenization is a manipulable variable, providing the basis for repairing glitch tokens at the tokenization level.

2.2 Glitch-Token Detection and Repair in LLMs

Glitch tokens are vocabulary entries that can induce anomalous behavior under certain inputs. When passed to downstream components such as retrieval or code execution, a single string-level error can cascade into workflow-wide failures (Figure 1). Prior work has linked glitch tokens to unsafe generation and behavior (Li et al., 2024; Tang et al., 2026), underscoring the necessity of repair.

Detection. Existing work identifies anomalous tokens through behavioral tasks. Early studies (Land & Bartolo, 2024; Wu et al., 2026) use repetition tasks to find tokens a model cannot reproduce correctly. Li et al. (2024) add spelling and length tasks, extending coverage to three dimensions. GlitchQuiz (Tang et al., 2026) further enriches prompt variants to improve robustness. In this work, we adopt the three shared task types as criteria: repetition (whether models can correctly repeat tokens), spelling (whether models can correctly spell tokens), and length (whether models can correctly count characters in tokens), and focus on repairing the identified glitch tokens.

Repair. Existing methods intervene internally. GlitchProber (Zhang et al., 2024) hooks into inference to correct intermediate activations; GlitchEdit (Tang et al., 2026) edits target embedding weights; GlitchCleaner (Fan et al., 2026) trains and loads a gated LoRA branch. All are effective in their settings but require access to model internals. For providers maintaining frozen checkpoints, every internal modification entails pipeline adjustments and potential regression risks. This paper investigates a stricter scenario: the base parameters and the forward implementation all remain unchanged, offering providers a more practical pathway to repair.

3 Empirical Study on Tokenization Repair

The previous section revealed the possibility of repairing glitch tokens externally by optimizing tokenization. The direct attempt along this direction is to delete the BPE merge rules that produce known glitch tokens so that the corresponding strings are encoded as their sub-tokens instead of the glitch IDs. This section conducts an empirical study of this approach and reports two key findings.

Setup. We select Llama-3.1-8B-Instruct (Meta, 2024) and Qwen3.8-27B (Qwen Team, 2026), representing the Llama BPE (128,256 entries) and Qwen BPE (248,320 entries) tokenizer families. Both use fixed checkpoints with BF16 precision and greedy decoding; Qwen uses text-only input with thinking disabled. We run the three GlitchQuiz (Tang et al., 2026) tasks (repetition, spelling, length) on every vocabulary entry; tokens passing all three are labeled normal, others glitch. Llama contains 22,026 glitch tokens (17.17%) and Qwen 29,451 (11.86%). The full protocol is in Appendix A.

Evaluation. For each glitch token, we reconstruct its merge tree from the BPE execution history and define three backtracking depths. **Depth-1** deletes only the rule producing the target token; **Depth-2**

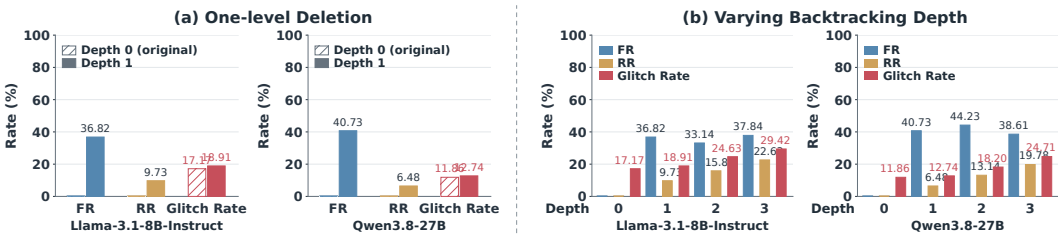

Figure 3. Repair gains and new errors from global merge deletion. (a) FR, RR, and GR after depth-1 backtracking on two models. (b) The three metrics at increasing backtracking depths.

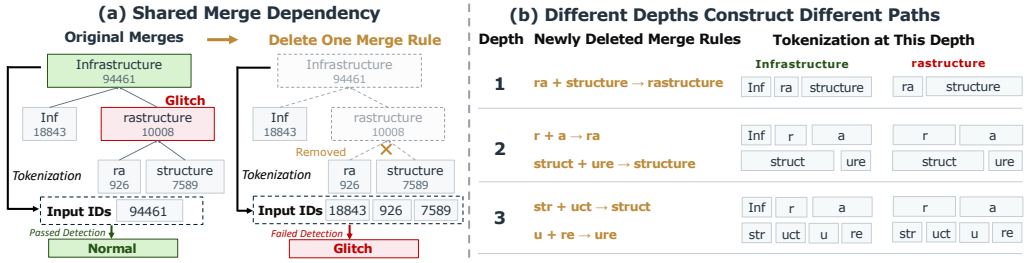

Figure 4. Collateral effects of shared merges and path variation across depths on Qwen3.8-27B. (a) Deleting the rule for the glitch token "rastructure" blocks the construction path of the normal token "Infrastructure", changing its encoding from one token to three and causing detection failure. (b) Different backtracking depths produce different sub-token paths for both tokens.

additionally deletes the rules producing its children; **Depth-3** extends one further level. At each depth, the deletion set is the union over all targets, applied globally. After deletion, we preserve the remaining rule priorities and vocabulary, re-encode all entries, and re-run detection.

Metrics. We use three metrics to measure repair effectiveness, all computed at the token level over the full vocabulary. ❶ Glitch Rate (GR) is the fraction of vocabulary tokens that still fail at least one of the three tests after repair, reflecting the net effect of repair gains and newly introduced errors. ❷ Fix Rate (FR) is the fraction of original glitch tokens that become normal. ❸ Regression Rate (RR) is the fraction of original normal tokens that fail at least one test after repair and thus become glitchy.

Results. Figure 3 shows FR, RR, and GR for both models at different backtracking depths. We analyze the results and report two findings.

Finding 1: Deleting glitch-token merge rules helps repair but introduces new errors. At depth-1, FR reaches 36.82% and 40.73% for Llama and Qwen, showing that changing the tokenization alone can fix a substantial fraction of failures (Figure 3a). However, RR reaches 9.73% and 6.48%, and GR increases (17.17% → 18.91% and 11.86% → 12.74%). The cause is that a glitch token can also serve as an intermediate merge node for a normal longer token. As shown in Figure 4(a), deleting the rule for "rastructure" prevents "Infrastructure" from forming through its original path, forcing it into three sub-tokens and causing detection failure. Global deletion cannot distinguish a target appearing as a final input from one used only during construction.

Finding 2: Fix rates vary by path while collateral damage grows with depth. Different depths essentially generate different sub-token sequences for the same glitch token (Figure 4b), and models differ in how well they handle each path. As shown in Figure 3(b), Llama's FR drops from 36.82% at depth-1 to 33.14% at depth-2, then rises to 37.84% at depth-3; Qwen exhibits the opposite trend, peaking at 44.23% at depth-2 and falling to 38.61% at depth-3. Meanwhile, RR rises monotonically with depth for both models, and the overall GR continues to worsen, indicating that collateral damage to normal tokens from global rule deletion is unavoidable and intensifies with depth.

These findings show that tokenization-level repair is viable, but global rule deletion cannot solve the problem: fix rates vary across paths with no universally optimal depth, and collateral damage to normal tokens is unavoidable. An effective repair must therefore find the best replacement path for each token individually while avoiding the impact on normal tokens.

4 GLITCHPATCH

To address the problems in Section 3, we propose **GLITCHPATCH**, a repair framework based on local retokenization. GLITCHPATCH has two stages (Figure 5): ❶ the offline stage uses Behavioral Path Optimization (BPO) to find the optimal replacement sequence for each glitch token. Since candidates per token can be numerous and full evaluation is expensive, BPO first ranks with a lightweight proxy score and evaluates the top candidates; ❷ the online stage applies local retokenization, replacing only the IDs of target glitch tokens while leaving normal tokens untouched. Details as follows.

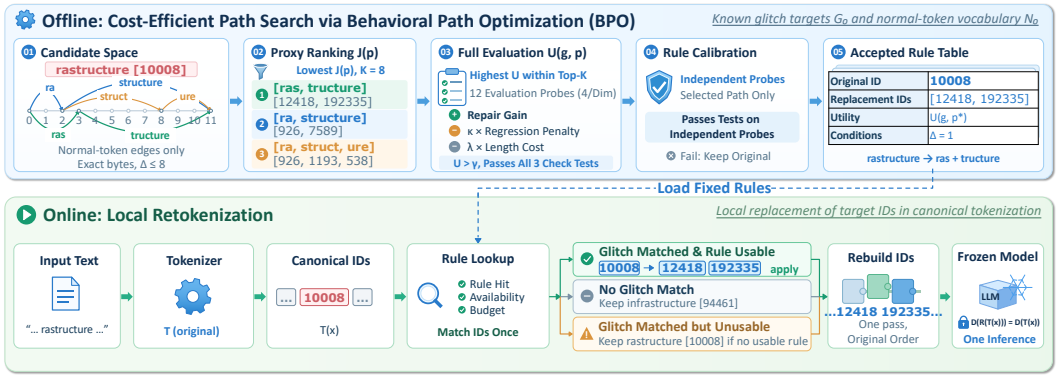

Figure 5. The two-stage GLITCHPATCH framework. The offline stage uses BPO to search for the behaviorally optimal replacement sequence for each glitch token and compiles validated results into a rule table. The online stage performs budget-constrained local ID substitution on the canonical tokenization output.

4.1 OFFLINE PATH SELECTION

Finding 2 shows that repair effectiveness varies by path and model, with no universally optimal strategy. Meanwhile, candidate paths per glitch token can be numerous, making exhaustive evaluation impractical. This subsection formalizes path selection as a constrained optimization problem and presents BPO to find the utility-maximizing replacement at manageable cost.

4.1.1 PROBLEM FORMULATION

Let T be the original tokenizer, R the patch transform, and x an input text. The patch must satisfy a decoding equivalence constraint:

$$D(R(T(x))) = D(T(x)), \tag{1}$$

ensuring that token ID sequences before and after replacement decode to the same bytes.

Candidate Space. For a glitch token g with byte sequence b, we treat each byte position as a node. If a normal token covers exactly $b[i:j]$, an edge links positions i and j. All start-to-end paths form $\mathcal{P}(g)$; every sub-token must belong to the normal set N_0.

Optimization Objective. We construct m evaluation probes (four prompt variants per dimension: repetition, spelling, length). Let m_+ be probes passed under the original representation, $N_{\text{fix}}(p)$ failed probes now passing, $N_{\text{reg}}(p)$ passed probes now failing, and $\Delta(p) = |p| - 1$ added tokens. The replacement utility is:

$$U(g,p) = \underbrace{\frac{N_{\text{fix}}(p)}{m}}_{\text{repair gain}} - \underbrace{\kappa \cdot \frac{N_{\text{reg}}(p)}{m_+}}_{\text{regression penalty}} - \underbrace{\lambda \cdot \frac{\Delta(p)}{B}}_{\text{length cost}}. \tag{2}$$

Here $\kappa > 1$ makes breaking a previously passed probe more costly than fixing a failed one; B is the length budget and λ controls length sensitivity. Eq. (2) is evaluated only when $m_+ > 0$; otherwise the original representation is retained. The offline goal is:

$$p^\star = \underset{p \in \mathcal{P}(g),\, U(g,p) > \gamma}{\arg\max} \; U(g,p) \quad \text{s.t. } D(p) = D(g),\; \Delta(p) \le B, \tag{3}$$

where γ ensures only candidates with positive net benefit are accepted. When no candidate qualifies, the original representation is retained.

4.1.2 BEHAVIORAL PATH OPTIMIZATION (BPO)

BPO solves Eq. (3) via a four-stage pipeline: construct the replacement space, rank candidates with a proxy score, perform full evaluation on the top-K paths, and confirm rules through independent calibration. Pseudocode is in Appendix C.5 (Algorithm 1) and the example in Figure 7.

Candidate Construction. BPO constructs $\mathcal{P}(g)$ as defined in Section 4.1.1. The vocabulary contributes tokens beyond the original merge tree, covering all valid equivalent paths. When no valid path exists, the original representation is retained. Graph-search properties are in Appendix C.3.

Proxy Ranking. Computing $U(g, p)$ requires heavy model calls, so BPO uses a proxy score to rank candidates and sends only the top-K to full evaluation. Because all components in N_0 pass the three base classification tests, those binary labels alone cannot distinguish quality. We estimate empirical failure rates $\hat{f}(v)$ and $\hat{f}(u, v)$ using three supplemental probes disjoint from all other prompts. The proxy score for path p=$(v_1, \ldots, v_k)$ is

$$J(p) = \sum_{j=1}^{|p|} \hat{f}(v_j) + \eta \sum_{j=2}^{|p|} \hat{f}(v_{j-1}, v_j) + \lambda_r \cdot \Delta(p). \tag{4}$$

Lower $J(p)$ indicates a more promising candidate. We default to η=0.5, λ_r=0.02, K=8, and B=8.

Full-sequence Evaluation. BPO inserts each shortlisted path into complete test inputs, using $m = 12$ selection probes to compute $U(g, p)$. A candidate must (i) pass all three check tests spanning three behavioral dimensions, whose templates are distinct from the GlitchQuiz prompts used for the reported FR/RR/GR; (ii) satisfy $U(g, p)$>γ. We default to κ=2, λ=0.02, and γ=0.05 (Appendix D.2).

Rule Confirmation. An independent set of calibration probes performs final confirmation. A rule is accepted only when all calibration probes pass; failure retains the original representation without testing the next-best candidate, preventing indirect data leakage. Each saved rule records the original ID, replacement IDs, content, added tokens, and utility.

4.2 ONLINE LOCAL RETOKENIZATION

Overall Flow. As in Figure 5, the online stage proceeds in four steps. First, the original tokenizer encodes the input text into canonical IDs $T(x)$. Second, the system performs a single-pass lookup against the offline rule table, checking each token ID for three conditions: whether it is a registered glitch token, whether a validated replacement rule exists, and whether the length budget permits substitution. Three cases arise: ❶ if all conditions are met, the glitch-token ID is replaced by its optimized sub-token sequence; ❷ if the token is not a glitch token, it passes through unchanged; ❸ if the token is a known glitch token but no usable rule exists or the budget is exceeded, the original ID is retained. Third, all tokens are reassembled in their original order to form the patched sequence. Finally, the patched sequence is fed to the frozen model for inference, satisfying $D(R(T(x))) = D(T(x))$. Because only identified glitch-token IDs are ever replaced, the IDs of unmatched normal tokens are preserved by construction (Appendix C.2).

Budget Allocation and Reconstruction. Each target may add at most B=8 tokens. When multiple targets co-occur, all applicable rules are applied to their corresponding positions and reconstructed in original order, with the length budget enforced independently per target. Rule matching is performed once against the initial canonical IDs and does not trigger recursively. The reconstructed sequence enters the model normally; no model parameters are changed.

5 EXPERIMENTAL SETUP

Having described the GLITCHPATCH framework, we now detail the models, evaluation protocol, metrics, and baseline configurations used to assess its effectiveness.

Models and Execution. We evaluate on ten models spanning six BPE tokenizer families with vocabularies from 32K to 262K: Llama-3.1-8B/70B-Instruct (Meta, 2024), Gemma-4-31B/E4B-it (Google, 2026), Qwen3.8-27B, Qwen3.6-27B, Qwen3.6-35B-A3B (Qwen Team, 2026), Mistral-7B-Instruct-v0.2 (Mistral AI, 2024), GLM-4.7-Flash (Zai-org, 2026), and Deepseek-vl2 (DeepSeek

AI, 2025) (text-only input). All use BF16, official chat templates, and greedy decoding. Per-model configurations are in Appendix Table A2; tokenizer adaptation details are in Appendix C.4.

Detection and Evaluation. For each model, every vocabulary token is tested on three GlitchQuiz tasks (repetition, spelling, length). Tokens passing all three are labeled normal; those failing any are labeled glitch. This full-vocabulary protocol eliminates selection bias and enables precise tracking of each token's status change. Scoring rules and sample sizes are in Appendix A.

Baselines. We compare against activation-based GlitchProber (Zhang et al., 2024) and parameter-based GlitchEdit (Tang et al., 2026) and GlitchCleaner (Fan et al., 2026). Each retains its required internal access; GLITCHPATCH's input-only restriction is not imposed on baselines. All methods share the same vocabulary detection results and classification criteria. Configurations are in Appendix D.3.

Repair Effectiveness. Following Section 3, we evaluate repair effectiveness using three metrics: ❶ Glitch Rate (GR), the fraction of vocabulary tokens that fail at least one of the three GlitchQuiz tasks; ❷ Fix Rate (FR), the fraction of original glitch tokens successfully repaired; and ❸ Regression Rate (RR), the fraction of original normal tokens that become glitches after repair.

Practicality Assessment. We assess practicality from these aspects: ❶ Time Cost: offline build time and online latency on 8×A100; ❷ Language Understanding: relative cross-entropy loss reduction on GPT-5.6-Terra-generated passages containing repaired tokens; ❸ General Capability: MMLU-Pro (Wang et al., 2024), ARC-Challenge (Bhakthavatsalam et al., 2021), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), with General Score as the arithmetic mean of four accuracies; ❹ We also evaluate contextual regression and out-of-distribution downstream performance (Appendix H).

6 EXPERIMENTAL RESULTS

We first report repair effectiveness in Section 6.1, followed by practicality evaluation, proxy-score analysis, repairability boundaries, and ablation studies in Sections 6.2–6.5.

6.1 REPAIR EFFECTIVENESS

Table 1 reports GLITCHPATCH's results on ten models. The mean FR is 85.10%, RR is 0.00%, and mean GR decreases from 14.88% to 2.27%. The highest FR is 91.04% (Llama-3.1-70B) and the lowest 80.50% (Gemma-4-E4B, largest vocabulary at 262K with 18.42% initial GR). The 0.00% RR follows from two factors: untriggered requests preserve original IDs by construction (Proposition 2), and in full-vocabulary testing each token is evaluated in isolation. Per-model counts are in Appendix B.

Table 1. GLITCHPATCH repair results across models. Full-vocabulary, token-level evaluation. Red subscripts show GR changes from the original model in percentage points.

Model	Glitch	Fixed	Reg.	FR% ↑	RR% ↓	GR% ↓
Llama-3.1-8B	22,026	18,722	0	85.00	0.00	$2.58_{-14.60}$
Llama-3.1-70B	15,878	14,455	0	91.04	0.00	$1.11_{-11.27}$
Gemma-4-31B	36,045	32,080	0	89.00	0.00	$1.51_{-12.24}$
Gemma-4-E4B	48,287	38,873	0	80.50	0.00	$3.59_{-14.83}$
Qwen3.8-27B	29,451	25,770	0	87.50	0.00	$1.48_{-10.38}$
Qwen3.6-27B	31,040	26,546	0	85.52	0.00	$1.81_{-10.69}$
Qwen3.6-35B-A3B	35,261	29,438	0	83.49	0.00	$2.34_{-11.85}$
Mistral-7B	6,240	5,083	0	81.46	0.00	$3.62_{-15.88}$
GLM-4.7	20,444	16,963	0	82.97	0.00	$2.25_{-10.95}$
Deepseek-vl2	20,426	17,267	0	84.53	0.00	$2.44_{-13.36}$
Mean				**85.10**	**0.00**	**2.27**$_{-12.61}$

Table 2 compares four methods. GLITCHPATCH's mean FR of 85.10% exceeds GlitchCleaner (70.73%) by 14.37 pp, GlitchEdit (58.39%) by 26.71 pp, and GlitchProber (24.63%) by 60.47 pp, achieving the highest FR on every model. All baselines require model-internal access, whereas GLITCHPATCH operates only on input IDs with fixed hyperparameters across six tokenizer families.

Table 2. Fix Rate comparison across four methods (% ↑). Green subscripts show GLITCHPATCH's gains over each baseline in percentage points.

Model	GlitchProber	GlitchEdit	GlitchCleaner	GLITCHPATCH
Llama-3.1-8B	21.29$_{+63.71}$	59.16$_{+25.84}$	68.07$_{+16.93}$	**85.00**
Llama-3.1-70B	30.97$_{+60.07}$	65.03$_{+26.01}$	78.48$_{+12.56}$	**91.04**
Gemma-4-31B	36.19$_{+52.81}$	66.37$_{+22.63}$	81.28$_{+7.72}$	**89.00**
Gemma-4-E4B	12.35$_{+68.15}$	51.77$_{+28.73}$	59.12$_{+21.38}$	**80.50**
Qwen3.8-27B	27.01$_{+60.49}$	60.48$_{+27.02}$	74.02$_{+13.48}$	**87.50**
Qwen3.6-27B	30.09$_{+55.43}$	61.67$_{+23.85}$	76.06$_{+9.46}$	**85.52**
Qwen3.6-35B-A3B	17.57$_{+65.92}$	52.82$_{+30.67}$	64.62$_{+18.87}$	**83.49**
Mistral-7B	20.96$_{+60.50}$	53.04$_{+28.41}$	67.45$_{+14.01}$	**81.46**
GLM-4.7	35.69$_{+47.28}$	62.83$_{+20.14}$	77.59$_{+5.38}$	**82.97**
Deepseek-vl2	14.22$_{+70.31}$	50.77$_{+33.76}$	60.59$_{+23.94}$	**84.53**
Mean	24.63$_{+60.47}$	58.39$_{+26.71}$	70.73$_{+14.37}$	**85.10**

6.2 PRACTICALITY ASSESSMENT

We select three representative models (Mistral-7B, Qwen3.8-27B, and Llama-3.1-70B) and compare all baseline methods. Figure 6 summarizes the results, with additional details in Appendices E–G.

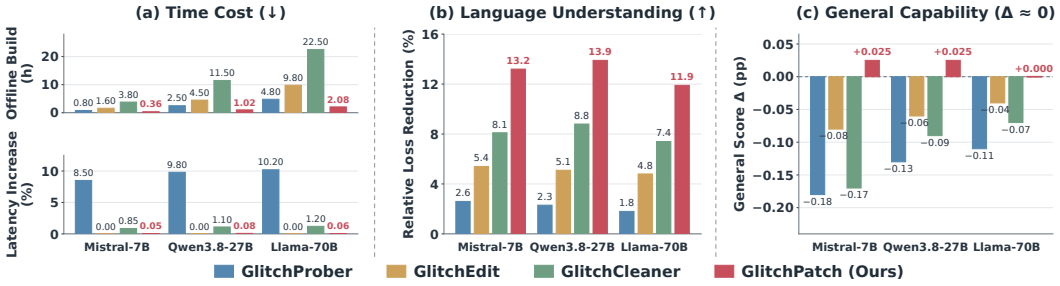

Figure 6. Practicality comparison across three representative models. (a) Offline build time and online latency increase. (b) Relative reduction in token-level loss. (c) Change in General Score.

Time Cost. We compare offline build time and online latency on 8×A100. GlitchProber computes activation statistics (0.80–4.80 h) and monitors activations at inference, incurring 8.50–10.20% online overhead. GlitchEdit searches for replacement embeddings (1.60–9.80 h) with no online cost. GlitchCleaner trains gated LoRA parameters with the highest offline cost (3.80–22.50 h) and adds 0.85–1.20% online latency. GLITCHPATCH builds in 0.36–2.08 h and introduces rule matching and sequence reconstruction with a small prefill cost, keeping the mean online latency increase < 0.1%.

Language Understanding. We sample 1,000 repaired glitch tokens per model, generate natural passages containing them via GPT-5.6-Terra, and compare token-level loss before and after repair. GLITCHPATCH's relative loss reductions are 13.2%, 13.9%, and 11.9% on the three models, exceeding all baselines. Replacing glitch tokens with well-learned sub-token sequences enables more accurate contextual prediction. The ten-model mean is 13.47% (Appendix G).

General Capability. On four benchmarks, GLITCHPATCH's General Score changes range from 0.000 to +0.025 pp on the three models, indicating no degradation. All baselines show small decreases (−0.04 to −0.18 pp) due to internal interventions affecting non-target tokens. Since GLITCHPATCH leaves input IDs unchanged for glitch-free inputs, capability is preserved (Appendix F).

Contextual and Downstream Evaluation. Beyond token-level metrics, we examine whether repair disrupts surrounding normal-token processing and whether its benefits transfer beyond the three GlitchQuiz dimensions. For contextual regression, we embed repairable glitch tokens in natural sentences and measure short-answer QA accuracy on adjacent normal-token content; GLITCHPATCH changes accuracy by at most ±0.2 pp across the tested models. A supplementary evaluation on uniformly sampled G_0 tokens yields the same conclusion. For out-of-distribution evaluation, we uniformly sample glitch tokens from G_0 and evaluate entity QA, verbatim copy and code-identifier completion under all four methods. GLITCHPATCH achieves the highest accuracy. Full protocols and results are reported in Appendix H.

6.3 Proxy Score Quality and Repairability

For each target with at least two candidates in a diagnostic set, we compute Spearman's ρ_g between $-J(p)$ and $U(g,p)$. The mean $\rho(-J,U)$ values are 0.72, 0.68, and 0.71 for Llama-8B, Qwen3.8, and Gemma-31B, confirming that lower-J candidates reliably yield higher U. Increasing K from 8 to 16 improves FR by only 0.43 pp while nearly doubling cost (Appendix D.5).

6.4 Repairability Analysis

Not all glitch tokens can be repaired. Table 3 analyzes Qwen3.8-27B's 29,451 glitch tokens by available path count. 5.00% have no legal equivalent path and are structurally unrepairable (short base units or byte-level tokens). Path count and FR correlate positively: 85.4% for 1–3 paths, 93.8% for 4–10, and 98.7% for >50. The largest group (4–10 paths, 39.87%) achieves 93.8% FR, indicating most glitch tokens have sufficient replacement options.

Table 3. Repairability analysis on Qwen3.8-27B.

Paths	Count	Proportion%	FR%
0	1,473	5.00	0.0
1–3	8,847	30.04	85.4
4–10	11,742	39.87	93.8
11–50	5,849	19.86	97.1
>50	1,540	5.23	98.7

6.5 Ablation Studies

We examine the effect of the key BPO parameter K (number of candidate paths retained per target) on repair effectiveness using Qwen3.8-27B. Table 4 shows FR and corresponding build cost at different K values. Build cost is estimated as $C(K) = C_{\text{shared}} + N_{\text{target}} \cdot K$.

C_{shared} denotes fixed preparation overhead (supplemental probe evaluations for the shared evidence pool) and $N_{\text{target}} \cdot K$ represents the total candidate evaluations.From $K = 1$ to $K = 8$, FR increases from 71.83% to 87.50%, with successive gains of 5.63, 5.45, and 4.59 pp. Further increasing to $K = 16$ yields only 0.43 pp additional gain while nearly doubling the build cost. This shows diminishing returns, so we adopt $K = 8$ as the default balancing repair effectiveness and evaluation budget.

Table 4. Candidate count K on Qwen3.8-27B.

K	FR (%)	Est. build cost
1	71.83	∼47,000
2	77.46	∼76,000
4	82.91	∼135,000
8	87.50	∼253,000
16	87.93	∼488,000

Table 5 ablates individual components. Restricting search to the BPE merge history reduces FR to 73.82% (−13.68 pp), showing the value of vocabulary-wide paths. Removing the pair term (η=0) lowers FR to 81.73%, confirming the benefit of adjacent-token compatibility, while removing full-path verification yields 85.34%, a further 2.16 pp drop. Setting γ=0 raises FR to 90.23% but admits marginal paths prone to regressions, so the default γ=0.05 trades 2.73 pp for greater stability. Overall, all four components contribute to repair quality. Complete ablations are in Appendix D.4.

Table 5. Component ablation on Qwen3.8-27B.

Setting	FR (%)	Note
Full GlitchPatch	87.50	
BPE merge history only	73.82	No out-of-tree paths
Remove pair term	81.73	Proxy ignores adjacency
Remove full-path evaluation	85.34	Proxy score only
Lower utility threshold ($\gamma = 0$)	90.23	Lower behavioral stability

7 Conclusion

This paper proposes GlitchPatch, a glitch-token repair framework that requires no access to model internals. By converting repair into local ID substitution through retokenization, GlitchPatch achieves 0.00% regression on normal tokens while reducing the mean glitch rate from 14.88% to 2.27% across ten models, outperforming all baselines with negligible overhead. We hope this work provides a practical option for improving tokenizer reliability.

AI USE STATEMENT

We used generative AI tools to assist with manuscript editing and language polishing, including improving grammar, clarity, and readability. The experimental protocol also uses GPT-5.6-Terra to generate target-containing passages for the loss and contextual-regression evaluations and to judge response relevance in the Entity QA evaluation. All AI-assisted revisions were reviewed and edited by the authors to preserve the intended scientific meaning and ensure consistency with the methods, experimental results, and conclusions. We take full responsibility for the final content of this work, including all text and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work repairs anomalous tokenizer behavior in publicly released language models without modifying model weights. All experiments use open-weight models and publicly available benchmarks. GLITCHPATCH operates as a lightweight input-level patch and does not introduce new training data or alter model capabilities beyond restoring intended tokenization. We do not foresee negative societal impacts specific to this work.

REPRODUCIBILITY STATEMENT

Appendix A specifies the detection and scoring protocol; Appendix B reports complete repair counts; and Appendices C–H provide implementation details, runtime, benchmark results, loss diagnostics, contextual regression, and downstream evaluations. Code and data will be released upon publication.

REFERENCES

Sumithra Bhakthavatsalam, Daniel Khashabi, Tushar Khot, Bhavana Dalvi Mishra, Kyle Richardson, Ashish Sabharwal, Carissa Schoenick, Oyvind Tafjord, and Peter Clark. Think you have solved direct-answer question answering? try arc-da, the direct-answer AI2 reasoning challenge. *CoRR*, abs/2102.03315, 2021. URL https://arxiv.org/abs/2102.03315.

Mark Chen, Jerry Tworek, Heewoo Jun, and et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek AI. deepseek-vl2 model card, 2025. URL https://huggingface.co/deepseek-ai/deepseek-vl2.

Yibo Fan, Jingru Li, and Huan Li. Glitchcleaner: Lightweight glitch tokens repairing by lossless gated lora in large language models. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 30656–30664. AAAI Press, 2026. doi: 10.1609/AAAI.V40I36.40321. URL https://doi.org/10.1609/aaai.v40i36.40321.

Poulami Ghosh and Preethi Jyothi. Language models are not equally robust to non-canonical tokenization across languages. *CoRR*, abs/2607.26831, 2026. doi: 10.48550/ARXIV.2607.26831. URL https://doi.org/10.48550/arXiv.2607.26831.

Google. Gemma 4 official model cards: 31B-it and E4B-it, 2026. URL https://huggingface.co/collections/google/gemma-4.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July*

15-20, 2018, Volume 1: Long Papers, pp. 66–75. Association for Computational Linguistics, 2018. doi: 10.18653/V1/P18-1007. URL https://aclanthology.org/P18-1007/.

Sander Land and Max Bartolo. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 11631–11646. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.649. URL https://doi.org/10.18653/v1/2024.emnlp-main.649.

Yuxi Li, Yi Liu, Gelei Deng, Ying Zhang, Wenjia Song, Ling Shi, Kailong Wang, Yuekang Li, Yang Liu, and Haoyu Wang. Glitch tokens in large language models: Categorization taxonomy and effective detection. *Proc. ACM Softw. Eng.*, 1(FSE):2075–2097, 2024. doi: 10.1145/3660799. URL https://doi.org/10.1145/3660799.

Meta. Llama 3.1 official model cards: 8B-Instruct and 70B-Instruct, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. The 70B checkpoint is available under the same organization as Llama-3.1-70B-Instruct.

Mistral AI. Mistral-7B-Instruct-v0.2 model card, 2024. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. Bpe-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 1882–1892. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.ACL-MAIN.170. URL https://doi.org/10.18653/v1/2020.acl-main.170.

Qwen Team. Qwen3.6 and Qwen3.8 model cards, 2026. URL https://huggingface.co/Qwen/collections. Checkpoints: Qwen3.8-27B, Qwen3.6-27B, and Qwen3.6-35B-A3B.

Kunsheng Tang, Peigui Qi, Yide Song, Wenbo Zhou, Zhicong Huang, Qing Guo, Tianwei Zhang, Weiming Zhang, Nenghai Yu, and Jie Zhang. One bad token spoils the barrel: Assessment, detection, and remediation of glitch tokens in large language models. In *35th USENIX Security Symposium (USENIX Security 26)*, pp. 2127–2146, Baltimore, MD, 2026. USENIX Association. URL https://www.usenix.org/conference/usenixsecurity26/presentation/tang-kunsheng.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Zihui Wu, Haichang Gao, and et al. Glitchminer: Mining glitch tokens in large language models via gradient-based discrete optimization. In *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 33998–34005. AAAI Press, 2026. doi: 10.1609/AAAI.V40I40.40693.

Zai-org. GLM-4.7-Flash model card, 2026. URL https://huggingface.co/zai-org/GLM-4.7-Flash.

Zhibo Zhang, Wuxia Bai, Yuxi Li, Mark Huasong Meng, Kailong Wang, Ling Shi, Li Li, Jun Wang, and Haoyu Wang. Glitchprober: Advancing effective detection and mitigation of glitch tokens in large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, pp. 643–655. ACM, 2024. doi: 10.1145/3691620.3695060.

Brian Siyuan Zheng, Alisa Liu, and et al. Broken tokens? your language model can secretly handle non-canonical tokenizations. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

A Data Construction, Evaluation Protocol, and Scoring

A.1 Glitch Token Detection Protocol

Each vocabulary token is tested on the three GlitchQuiz dimensions: repetition, spelling, and length. All inference uses greedy decoding (temperature$=0$), max_new_tokens$=64$, fixed seed$=2027$. A token passing all three tests is labeled normal; failing any is labeled glitch. This is a binary classification with no partial-glitch state. The scoring rules are defined below. Each token is evaluated on the three GlitchQuiz tasks described in Table A1.

Table A1. The three GlitchQuiz tasks and their scoring rules.

Dimension	Instruction	Expected output	Scoring criterion
Repetition	Repeat the string: {target}	The target itself	Exact byte match
Spelling	Spell character by character: {target}	Character sequence	Unicode character order match
Length	Count the characters: {target}	Correct count	Extracted number matches Unicode char count

A.2 Byte-Level and Control-Token Boundaries

For tokens containing incomplete UTF-8 sequences (e.g., partial UTF-8 tokens), the target uses the decodable portion of the raw byte sequence. If the byte sequence cannot be decoded into a valid Unicode string, spelling and length tasks operate at the byte level: spelling requires byte-by-byte enumeration, and length requires the correct byte count.

A.3 Vocabulary Sizes and Initial Glitch Rates

Table A2 shows the vocabulary testing scale for each model. Glitch ratios vary across models. Mistral-7B has the highest glitch ratio (19.50%) and Qwen3.8-27B the lowest (11.86%). This difference mainly stems from vocabulary size, training corpus coverage, and tokenizer training strategy. Mistral-7B has the smallest vocabulary yet the highest glitch ratio, indicating that glitch ratio and vocabulary size are not simply proportional. Gemma-4-E4B has the largest vocabulary.

Table A2. Vocabulary sizes and initial glitch counts per model.

Model	Vocab	Glitch	Glitch %	Normal
Llama-3.1-8B	128,256	22,026	17.17	106,230
Llama-3.1-70B	128,256	15,878	12.38	112,378
Gemma-4-31B	262,144	36,045	13.75	226,099
Gemma-4-E4B	262,144	48,287	18.42	213,857
Qwen3.8-27B	248,320	29,451	11.86	218,869
Qwen3.6-27B	248,320	31,040	12.50	217,280
Qwen3.6-35B-A3B	248,320	35,261	14.20	213,059
Mistral-7B	32,000	6,240	19.50	25,760
GLM-4.7	154,880	20,444	13.20	134,436
Deepseek-vl2	129,280	20,426	15.80	108,854

A.4 Decoding and Execution Settings

All detection, selection/calibration probes, and repair evaluations use greedy decoding (temperature$=0$, top_$k=1$), maximum generation length of 64 tokens, BF16 precision, and fixed seed 2027. The fixed seed and greedy decoding jointly ensure deterministic output for identical inputs on the same model, making token-level normal/glitch classification fully reproducible across runs. For model-specific settings: Llama-3.1 series, Gemma-4 series (text-only mode), Mistral-7B, and GLM-4.7 all use their respective official chat templates; Qwen3.8-27B and Qwen3.6 series set enable_thinking=False for text-only mode; deepseek-vl2 also uses text-only input mode.

A.5 SHARED EVIDENCE POOL.

The initial three-task classification assigns each token a binary normal/glitch label, but all components in N_0 pass all three tests and therefore share an identical label. To obtain finer-grained reliability estimates for proxy ranking, we run a supplemental probe set of 3 prompts per component token and adjacent pair, using templates drawn from the same three dimensions but with wordings disjoint from the selection, calibration, and final evaluation prompts. $\hat{f}(v)$ is the fraction of supplemental probes on which token v fails; $\hat{f}(u, v)$ is the fraction on which the adjacent pair (u, v) fails. Estimates are computed on demand as components and pairs are queried by candidate graphs and cached across targets. For a component or pair not yet queried by any target's graph, no estimate is needed because it does not appear in any candidate path.

B COMPLETE REPAIR RESULTS

B.1 AGGREGATE RESULTS ACROSS METHODS

The core design property of GLITCHPATCH is that it only replaces the IDs of known glitch tokens without touching normal-token encodings. Therefore, GLITCHPATCH's RR is 0.00% on all models: any token originally classified as normal retains its original encoding after GLITCHPATCH is applied, and the model receives input IDs identical to the original. This $\mathrm{RR} = 0.00\%$ is a structural property determined by the method design: GLITCHPATCH operates on the canonical tokenization output and only substitutes positions matching known glitch-token IDs in the rule table. Normal-token IDs are not in the rule table, so their encoding process and final IDs remain completely unchanged. This differs from the baselines: GlitchProber's activation intervention may affect subsequent token processing, GlitchEdit's embedding editing may affect the output head under weight tying, and GlitchCleaner's gated LoRA branch may produce small activations on non-target tokens.

Table B1. Aggregate repair results across all four methods.

Method	Fixed	Avg. FR(%)	Regressed	Avg. RR(%)	Avg. GR(%)
GlitchProber	63,614	24.63	382	0.024	11.37
GlitchEdit	154,382	58.39	182	0.012	6.30
GlitchCleaner	185,940	70.73	713	0.046	4.53
GLITCHPATCH	**225,197**	**85.10**	**0**	**0.00**	**2.27**

B.2 PER-MODEL COMPLETE COUNTS

Table B2 gives the complete per-model counts for each method. Reg. and Maint. denote regressed and maintained counts. FR, RR, and GR are percentages.

Table B2. Per-model complete counts for all four methods. Glitch and normal counts are in model heading.

Method	Fixed	Unfixed	FR%	Reg.	Maint.	RR%	GR%
Llama-8B $\lvert G_0\rvert = 22,026$; $\lvert N_0\rvert = 106,230$							
GlitchProber	4,689	17,337	21.29	32	106,198	0.03	13.54
GlitchEdit	13,031	8,995	59.16	11	106,219	0.01	7.02
GlitchCleaner	14,993	7,033	68.07	53	106,177	0.05	5.52
GLITCHPATCH	**18,722**	**3,304**	**85.00**	**0**	**106,230**	**0.00**	**2.58**
Llama-70B $\lvert G_0\rvert = 15,878$; $\lvert N_0\rvert = 112,378$							
GlitchProber	4,917	10,961	30.97	11	112,367	0.01	8.55
GlitchEdit	10,325	5,553	65.03	11	112,367	0.01	4.34
GlitchCleaner	12,461	3,417	78.48	34	112,344	0.03	2.69
GLITCHPATCH	**14,455**	**1,423**	**91.04**	**0**	**112,378**	**0.00**	**1.11**

Continued on the next page.

Table B2 (continued)

Method	Fixed	Unfixed	FR%	Reg.	Maint.	RR%	GR%
Gemma-31B $\lvert G_0\rvert = 36,045$; $\lvert N_0\rvert = 226,099$							
GlitchProber	13,045	23,000	36.19	45	226,054	0.02	8.79
GlitchEdit	23,923	12,122	66.37	23	226,076	0.01	4.63
GlitchCleaner	29,297	6,748	81.28	90	226,009	0.04	2.61
GLITCHPATCH	**32,080**	**3,965**	**89.00**	**0**	**226,099**	**0.00**	**1.51**
Gemma-E4B $\lvert G_0\rvert = 48,287$; $\lvert N_0\rvert = 213,857$							
GlitchProber	5,963	42,324	12.35	86	213,771	0.04	16.18
GlitchEdit	24,998	23,289	51.77	43	213,814	0.02	8.90
GlitchCleaner	28,547	19,740	59.12	128	213,729	0.06	7.58
GLITCHPATCH	**38,873**	**9,414**	**80.50**	**0**	**213,857**	**0.00**	**3.59**
Qwen3.8-27B $\lvert G_0\rvert = 29,451$; $\lvert N_0\rvert = 218,869$							
GlitchProber	7,956	21,495	27.01	44	218,825	0.02	8.67
GlitchEdit	17,813	11,638	60.48	22	218,847	0.01	4.70
GlitchCleaner	21,800	7,651	74.02	88	218,781	0.04	3.12
GLITCHPATCH	**25,770**	**3,681**	**87.50**	**0**	**218,869**	**0.00**	**1.48**
Qwen3.6-27B $\lvert G_0\rvert = 31,040$; $\lvert N_0\rvert = 217,280$							
GlitchProber	9,340	21,700	30.09	43	217,237	0.02	8.76
GlitchEdit	19,142	11,898	61.67	22	217,258	0.01	4.80
GlitchCleaner	23,609	7,431	76.06	87	217,193	0.04	3.03
GLITCHPATCH	**26,546**	**4,494**	**85.52**	**0**	**217,280**	**0.00**	**1.81**
Qwen3.6-A3B $\lvert G_0\rvert = 35,261$; $\lvert N_0\rvert = 213,059$							
GlitchProber	6,195	29,066	17.57	64	212,995	0.03	11.73
GlitchEdit	18,625	16,636	52.82	21	213,038	0.01	6.71
GlitchCleaner	22,786	12,475	64.62	107	212,952	0.05	5.07
GLITCHPATCH	**29,438**	**5,823**	**83.49**	**0**	**213,059**	**0.00**	**2.34**
Mistral-7B $\lvert G_0\rvert = 6,240$; $\lvert N_0\rvert = 25,760$							
GlitchProber	1,308	4,932	20.96	8	25,752	0.03	15.44
GlitchEdit	3,310	2,930	53.04	5	25,755	0.02	9.17
GlitchCleaner	4,209	2,031	67.45	15	25,745	0.06	6.39
GLITCHPATCH	**5,083**	**1,157**	**81.46**	**0**	**25,760**	**0.00**	**3.62**
GLM-4.7 $\lvert G_0\rvert = 20,444$; $\lvert N_0\rvert = 134,436$							
GlitchProber	7,296	13,148	35.69	27	134,409	0.02	8.51
GlitchEdit	12,845	7,599	62.83	13	134,423	0.01	4.91
GlitchCleaner	15,862	4,582	77.59	67	134,369	0.05	3.00
GLITCHPATCH	**16,963**	**3,481**	**82.97**	**0**	**134,436**	**0.00**	**2.25**
Deepseek-vl2 $\lvert G_0\rvert = 20,426$; $\lvert N_0\rvert = 108,854$							
GlitchProber	2,905	17,521	14.22	22	108,832	0.02	13.57
GlitchEdit	10,370	10,056	50.77	11	108,843	0.01	7.79
GlitchCleaner	12,376	8,050	60.59	44	108,810	0.04	6.26
GLITCHPATCH	**17,267**	**3,159**	**84.53**	**0**	**108,854**	**0.00**	**2.44**

C FORMAL PROPERTIES AND TOKENIZER ADAPTATION

C.1 LOCAL EQUIVALENCE (PROPOSITION 1)

Let the tokenizer adapter's local decoding transition be

$$\mathcal{D}(s, p) = (b, s'), \tag{5}$$

where s is the decoder state entering the span, p is the token sequence, b is the output byte content, and s' is the exit state. A replacement $p^0 \mapsto p$ is valid when

$$\mathcal{D}(s, p^0) = \mathcal{D}(s, p), \tag{6}$$

and the special-token ID sequences within both spans are identical.

Proposition 1 (Compatible local replacements preserve global content)**.** *For a set of non-overlapping replacements whose entry states match and whose local transitions satisfy decoding equivalence, the final sequence has the same decoded bytes, special-token ID sequence, and decoder exit state as the original sequence.*

Proof. Process spans in their original order. The first replacement preserves both its emitted bytes and exit state, so the subsequent unchanged segment receives the same entry state. Repeat the same argument for each later replacement. Induction establishes equality of the full output and terminal state. If adjacent spans cannot be validated independently, validate their joint region; without a joint rule, retain the original. □

This proposition covers implementations satisfying the adapter contract. Additional normalization, text cleanup, or control-structure handling must be included in the state or fixed before the transformation, and cannot change content outside the contract.

C.2 Preservation of Untriggered Inputs (Proposition 2)

Proposition 2 (Unchanged effective inputs preserve the model conditional distribution)**.** *If $R(T(x)) = T(x)$ and the model parameters, attention mask, position IDs, and decoding configuration are identical, then*

$$P_\theta(y \mid R(T(x))) = P_\theta(y \mid T(x)). \tag{7}$$

Proof. Both conditions send identical effective inputs to the same model, defining the same conditional distribution. Under identical deterministic execution, greedy outputs are also identical. □

Scope of the preservation guarantee. Proposition 2 directly explains the $\mathrm{RR} = 0.00\%$ result observed across all ten models in full-vocabulary testing. Since GlitchPatch's rule table contains only glitch-token IDs, when a normal token appears in the input it matches no rule, so $R(T(x)) = T(x)$ holds. The model receives input IDs identical to the original encoding, and the conditional distribution is unchanged. Baseline methods cannot provide the same guarantee because their repairs act on model internals (activations, embeddings, or adapter parameters), and internal modifications can propagate to non-target token processing through attention mechanisms, residual connections, or weight tying.

C.3 Graph Search Guarantee

Every ordinary edge in the candidate graph advances by at least one byte; empty-content edges and control tokens are excluded. Under fixed boundaries and finite decoder state, the graph is acyclic. Augmenting each node with the previous token and used length makes the pairwise term in Equation 4 additive, enabling budgeted top-K path search on the augmented graph without enumerating every complete path.

Let $|E'|$ denote the edge count after state augmentation. A conservative complexity bound for direct implementation is $O(BK|E'|\log K)$ when length is tracked separately; if E' already incorporates the length dimension, the B factor should not be counted again. Actual cost depends on trie matching and the number of reachable states. The vocabulary trie is built once and reused across targets.

The graph objective is a proposal cost, not the true $-\log P_\theta(p \mid c)$. Full model probabilities depend on the complete prefix and generally cannot be decomposed into fixed local edge costs on the same graph. Full-context behavioral evaluation is therefore necessary to assess shortlisted paths.

C.4 Tokenizer Adapter Contract

The evaluated models span two main BPE implementations: tiktoken-style byte-level BPE (Llama, Qwen, GLM, deepseek-vl2) and SentencePiece BPE (Gemma, Mistral). All implementations share the candidate graph construction logic but require separate adaptation for byte sequence extraction,

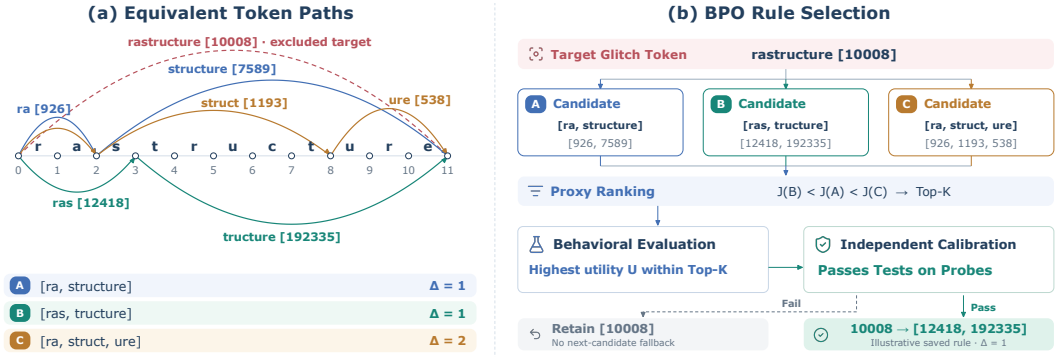

Figure 7. Candidate paths and rule selection. (a) Normal vocabulary entries define different paths over the same byte sequence. The original glitch ID is excluded from replacement edges. (b) Proxy ranking, full-sequence evaluation, and independent calibration select a rule.

Table C1. Tokenizer adapter targets per model.

Model	Type	Vocab	Adaptation
Llama-3.1-8B	Byte-level BPE	128,256	Raw byte graph
Llama-3.1-70B	Byte-level BPE	128,256	Shared with 8B
Gemma-4-31B	SentencePiece	262,144	Normalizer + state
Gemma-4-E4B	SentencePiece	262,144	Shared with 31B
Qwen3.8-27B	Byte-level BPE	248,320	Raw byte graph
Qwen3.6-27B	Byte-level BPE	248,320	Same as above
Qwen3.6-A3B	Byte-level BPE	248,320	Shared with 3.6-27B
Mistral-7B	SentencePiece	32,000	Normalizer + state
GLM-4.7	Byte-level BPE	154,880	Raw byte graph
Deepseek-vl2	Byte-level BPE	129,280	Raw byte graph

Table C2. Tokenizer adapter contract.

Case	Required handling	Invalid shortcut
Byte-level BPE	Build edges from exact raw bytes	Substitute visible Unicode for incomplete bytes
SentencePiece BPE	Preserve normalizer, leading space, byte fallback, state	Infer algorithm from file format alone
Partial UTF-8	Expand to complete verifiable span; compare joint decoding	Decode to replacement character and build from it
Added/special tokens	Preserve IDs, order, control structure; exclude as edges	Reconstruct control markers via string matching
Normalization	Use $T(x)$'s actual byte content as reference	Claim preservation without checking
Incompatible/over-budget	Retain canonical IDs	Delete chars, insert spaces, or rewrite text

decoder state management, and leading-space handling. Tables C1 and C2 summarize the adapter targets and contract.

C.5 BPO Pseudocode and Candidate Path Graph

Algorithm 1 summarizes the offline construction. Figure 7 illustrates three candidate paths for "rastructure"; the token IDs shown are tokenizer-specific.

Algorithm 1 Behavioral Path Optimization (BPO)

Input: tokenizer T; glitch set G_0; normal set N_0; budget B; shortlist size K; proxy parameters (η, λ_r); utility parameters $(\kappa, \lambda, \gamma)$; selection probes P_s; calibration probes P_c.
Output: replacement rule table Rules.

1. Initialize $\mathrm{Rules} \leftarrow \varnothing$ and an empty shared evidence cache $\mathcal{E}$.
2. **For each** $g \in G_0$:
   a. Evaluate the original representation on P_s to obtain m_+. If $m_+ = 0$, retain g and continue to the next target.
   b. Extract byte sequence b of g; build directed graph using N_0 tokens covering b.
   c. For each component token or adjacent pair required by the graph but absent from $\mathcal{E}$, estimate $\hat{f}(v)$ or $\hat{f}(u, v)$ using the supplemental probes and cache the result.
   d. Retrieve up to K paths with the smallest $J(p)$ subject to $\Delta(p) \leq B$ and the decoder contract.
   e. Initialize $\mathrm{best} \leftarrow \varnothing$, $U_{\mathrm{best}} \leftarrow \gamma$.
   f. **For each** shortlisted p: compute $U(g, p)$ on P_s. If p passes all three behavioral dimensions on the selection probes P_s and $U(g, p) > U_{\mathrm{best}}$, update $\mathrm{best} \leftarrow p$, $U_{\mathrm{best}} \leftarrow U(g, p)$.
   g. If best exists, run independent calibration on P_c.
   h. If all calibration probes pass, store the rule (original ID, replacement IDs, content, added length, utility). Otherwise retain g; do not test the next candidate on P_c.
3. **Return** Rules.

D Diagnostics, Parameters, and Additional Analyses

D.1 Selection Probe Structure

Table D1 describes the 12 selection probes: four prompt variants per GlitchQuiz dimension. Each variant uses a different question format and context structure to more thoroughly assess replacement robustness on that dimension. The calibration stage uses 4 independent new probes (1–2 per dimension) that do not participate in the preceding ranking or selection. All selection and calibration probes share the same decoding budget as Appendix A: greedy decoding, max_new_tokens$= 64$.

Table D1. Twelve selection probes: four variants per dimension. g denotes the target string.

Dim.	Variant	Prompt structure	Criterion
Rep.	Direct	Repeat g exactly	Exact match
	Embedded	Extract g from sentence	Exact extraction
	Label bind	Bind g to label A; query	Exact match
	Selection	Select g from list	Correct selection
Spell	Forward	List chars of g in order	Correct sequence
	Reverse	List chars in reverse	Correct reverse
	Locate	Give the kth char of g	Correct char
	Frequency	Count char c in g	Correct count
Length	Direct	Report length of g	Correct length
	Compare	Compare g vs. h	Correct choice
	Interval	Is length in $[a, b]$?	Correct judgment
	Enum+count	List chars then total	Both correct

D.2 Default Parameters

Table D2 lists all default parameters. Global parameters are selected on a Qwen3.8-27B development set from a fixed grid: $K \in \{1, 2, 4, 8, 16\}$, $B \in \{4, 8, 16\}$, $\eta \in \{0, 0.25, 0.50, 1.00\}$, $\kappa \in \{1, 2, 4\}$,

$\gamma \in \{0, 0.05, 0.10\}$. The optimization objective is to maximize net repair gain under limited sequence growth. The selected configuration is fixed for all other models and the main evaluation.

Table D2. Default GLITCHPATCH parameters.

Parameter	Value	Meaning
K	8	Max candidate paths per target
B	8	Max added tokens per span
Span content limit	64 bytes	Retain if exceeded
η	0.5	Adjacent-pair proxy weight
λ_r	0.02	Proxy length cost
κ	2	Regression penalty coefficient
λ	0.02	Utility length cost
γ	0.05	Minimum selection utility
Shared-pool probes	3/entry	For components and pairs
Selection / calib.	12 / 4	Per-target contexts
Probe output limit	64 tokens	Matching detection budget

D.3 BASELINE ADAPTATION CONFIGURATIONS

Table D3 describes the construction and adaptation settings for each baseline. GlitchCleaner uses AdamW with effective batch size 32, maximum sequence length 512, warmup ratio 0.03, and zero weight decay. GlitchEdit follows the model's actual parameter sharing when embeddings are tied, without silently untying them. GLITCHPATCH's rule table only changes input IDs, so the model's parameter sharing remains intact.

Table D3. Baseline construction and adaptation settings.

Method	Mechanism & config	Parameter selection
GlitchProber	Activation statistics + neuron correction; 2,000 ref. tokens; threshold 0.99; later MLP layers	Relative layer interval grid; $\sim[0.60L, 0.90L]$
GlitchEdit	Target embedding mixed with normal neighbors; $\alpha \in [0, 1]$, step 0.1; 1–10 neighbors	Behavioral acceptance; retain failure fallback
GlitchCleaner	Target-triggered MLP gated LoRA; rank 4, scaling 4, lr 3×10^{-4}	Freeze base weights; independent calib. from $\leq$20 epochs

D.4 PARAMETER AND COMPONENT ABLATIONS

The parameter sweeps in Tables D4–D6 use Qwen3.8-27B with $K = 8$. Ablation experiments in Tables D7 are conducted on three models using each model's full glitch set.

Utility threshold γ. Table D4 shows the effect of γ. Here, *coverage* denotes the fraction of the initial glitch set G_0 for which a replacement rule is stored after candidate selection and independent calibration, whereas FR is measured afterwards using the disjoint final GlitchQuiz evaluation. Coverage therefore upper-bounds FR but need not equal it: a stored rule may pass selection and calibration yet still fail to convert the target into a normal token under the final evaluation. Lowering γ only relaxes the utility threshold; the three check tests and independent calibration still apply. At $\gamma = 0$, coverage rises to 94.8%, close to the fraction of Qwen3.8-27B targets with at least one feasible path and $m_+ > 0$, and FR rises to 90.23%. The default $\gamma = 0.05$ yields 89.2% coverage and 87.50% FR; the approximately 1.7 pp difference corresponds to targets that receive a stored rule but do not pass the final evaluation. At $\gamma = 0.10$, the more conservative threshold reduces coverage to 81.8% and FR to 80.86%.

Table D4. Utility threshold γ.

γ	0.00	0.05	0.10
FR (%)	90.23	87.50	80.86
Coverage (%)	94.8	89.2	81.8

Adjacent-pair weight η. Table D5 shows the effect of η. At $\eta = 0$, adjacency compatibility is ignored, yielding 81.73% FR. The default $\eta = 0.50$ improves FR to 87.50%, indicating that moderate pairwise information helps distinguish more reliable replacement paths. At $\eta = 1.00$, however, excessive emphasis on pairwise risk can down-rank paths whose sub-tokens are individually better, reducing FR to 86.47%. This suggests that pairwise compatibility is useful as a complementary signal but should not dominate the path-level score.

Table D5. Adjacent-pair weight η.

η	0.00	0.25	0.50	1.00
FR (%)	81.73	85.03	87.50	86.47

Span token budget B. Table D6 shows the effect of B. At $B = 4$, some glitch tokens requiring finer decomposition cannot find effective paths within budget, yielding 82.48% FR. The difference between $B = 8$ (default) and $B = 16$ is only 1.02 pp, but the mean token increase rises from 2.6% to 4.1%, adding online sequence length cost, suggesting $B = 8$ provides a better balance.

Table D6. Span token budget B.

B	4	8	16
FR (%)	82.48	87.50	88.52
Mean growth (%)	1.8	2.6	4.1

The role of κ. κ controls the penalty strength for probe-level behavioral regression during path selection, which is distinct from the vocabulary-level RR metric. Because the gain and regression terms use different denominators, a regressed probe has $\kappa m/m_+$ times the weight of a repaired probe when $m_+ > 0$. Thus, at $\kappa = 1$, a regressed probe is at least as costly as a repaired probe; at $\kappa = 2$, it is at least twice as costly, steering the algorithm toward behaviorally stable paths. At $\kappa = 4$, the stronger penalty over-excludes candidates with positive net benefit and slightly reduces FR.

Three-model component ablation. Table D7 extends the component ablation to three models. All three show consistent trends: BPE-history-only search yields the lowest FR, removing the gate gives the highest classification FR but weaker behavioral guarantees, and the full method balances FR with reliability. Gemma-4-31B has slightly higher FR across all settings, possibly because its larger vocabulary provides more equivalent paths.

Table D7. Three-model component ablation (FR%).

Setting	Llama-8B	Qwen3.8	Gemma-31B
Full GLITCHPATCH	85.00	87.50	89.00
BPE history only	72.41	73.82	76.53
Remove pair term	79.27	81.73	84.18
Remove path eval.	82.86	85.34	87.12
Threshold $\gamma = 0$	88.16	90.23	91.74

D.5 PROXY SCORE QUALITY ANALYSIS

Table D8 reports the mean per-target Spearman correlation $\rho(-J, U)$. For each target with at least two candidates in the separately evaluated diagnostic set, the correlation is computed between $-J(p)$ and $U(g, p)$; targets with fewer than two evaluated candidates are excluded. The mean values range from 0.68 to 0.72, indicating that lower-J candidates tend to have higher utility. Increasing K from 8 to 16 improves FR by only 0.43 pp while nearly doubling candidate-evaluation cost, indicating diminishing returns beyond $K = 8$.

Table D8. Proxy score vs. true utility.

Model	Mean $\rho(-J, U)$
Llama-3.1-8B	0.72
Qwen3.8-27B	0.68
Gemma-4-31B	0.71

E RUNTIME AND COST

E.1 PER-MODEL SERVING LATENCY

Table E1 shows GLITCHPATCH's online latency and offline build time for each model. GLITCHPATCH's online overhead comprises two parts: lightweight host-side rule matching and sequence

reconstruction, and the increased prefill computation caused by the additional input tokens on triggered requests. The language-model inference itself remains GPU-based. Since the added tokens affect only the prefill stage and do not increase the number of decoded output tokens, the mean end-to-end latency increase across models is only 0.065%, well below 0.1%. Offline build times range from 0.36 h (Mistral-7B) to 2.08 h (Llama-3.1-70B), with a mean of 0.89 h.

Table E1. GLITCHPATCH per-model serving latency and offline build time.

Model	TP	Base (s)	Incr. (%)	Host Prep. (ms)	Build (h)
Llama-3.1-8B	1	1.280	0.071	0.231	0.61
Llama-3.1-70B	4	4.600	0.058	0.247	2.08
Gemma-4-31B	2	2.950	0.069	0.258	1.27
Gemma-4-E4B	1	1.100	0.061	0.219	0.49
Qwen3.8-27B	2	2.750	0.079	0.243	1.02
Qwen3.6-27B	2	2.800	0.068	0.238	0.96
Qwen3.6-A3B	2	1.850	0.062	0.214	0.74
Mistral-7B	1	0.950	0.051	0.187	0.36
GLM-4.7	1	1.500	0.059	0.221	0.56
Deepseek-vl2	2	2.200	0.072	0.229	0.84
Mean	–	–	**0.065**	**0.229**	**0.89**

E.2 BUILD TIME COMPARISON

Table E2 compares the offline build times of all four methods across ten models. GLITCHPATCH's average build time is 0.89 h, approximately 41% of GlitchProber (2.19 h), 21% of GlitchEdit (4.28 h), and 9% of GlitchCleaner (10.41 h). GlitchCleaner requires training gated LoRA parameters and has the longest build time, reaching 22.50 h on Llama-70B. GlitchProber's build time includes computing activation statistics for reference tokens.

Table E2. Offline build time comparison across methods (hours).

Model	Prober	Edit	Cleaner	GLITCHPATCH
Llama-3.1-8B	0.80	2.80	7.20	**0.61**
Llama-3.1-70B	4.80	9.80	22.50	**2.08**
Gemma-4-31B	2.80	5.20	12.80	**1.27**
Gemma-4-E4B	2.00	4.00	9.80	**0.49**
Qwen3.8-27B	2.50	4.50	11.50	**1.02**
Qwen3.6-27B	2.40	4.30	11.00	**0.96**
Qwen3.6-A3B	2.00	3.80	9.20	**0.74**
Mistral-7B	0.80	1.60	3.80	**0.36**
GLM-4.7	1.60	3.00	7.50	**0.56**
Deepseek-vl2	2.20	3.80	8.80	**0.84**
Mean	2.19	4.28	10.41	**0.89**

E.3 COMBINED COST SUMMARY

Table E3 summarizes the offline and online costs of all four methods. GLITCHPATCH has the shortest average offline build time (0.89 h) while keeping the mean online latency increase below 0.1%. It requires no additional model parameters or extra model calls. GlitchProber has the highest online latency increase ($\sim$9.20%) because every inference requires activation monitoring and correction. GlitchEdit introduces no additional serving latency after editing, but requires a substantially longer offline build stage. GlitchCleaner has the longest build time and introduces $\sim$1.00% additional latency from the adapter branch.

Table E3. Combined cost summary (ten-model average).

Method	Build (h)	Latency incr.	Extra params	Extra calls
GlitchProber	2.19	∼9.20%	0	0
GlitchEdit	4.28	0.00%	0	0
GlitchCleaner	10.41	∼1.00%	Gated LoRA	0
GLITCHPATCH	**0.89**	**<0.1%**	**0**	**0**

F General Capability and Repair Boundaries

F.1 General Benchmarks

Table F1 shows the correct-answer counts of tested models on four general benchmarks (MMLU-Pro 1,000 items, ARC-Challenge 1,000 items, GSM8K 1,000 items, HumanEval 164 items). GLITCHPATCH and the original models use identical test samples and decoding settings. Across models and tasks, the correct counts remain nearly unchanged, with small sample-level fluctuations, indicating that local ID replacement has negligible aggregate impact on general performance. Any observed difference can arise on benchmark instances whose canonical tokenization contains at least one rule-matched glitch-token ID; rule-unmatched instances remain exactly unchanged by construction.

Table F1. Original vs. GLITCHPATCH correct counts on general benchmarks.

Model	Method	MMLU-Pro	ARC	GSM8K	HumanEval
Llama-8B	Original	540	790	805	86
	GLITCHPATCH	540	791	805	86
Llama-70B	Original	660	875	922	121
	GLITCHPATCH	660	875	922	121
Gemma-31B	Original	640	860	910	118
	GLITCHPATCH	641	860	910	118
Gemma-E4B	Original	480	740	760	72
	GLITCHPATCH	479	740	760	72
Qwen3.8	Original	630	850	900	115
	GLITCHPATCH	631	850	900	115
Qwen3.6-27B	Original	620	845	895	113
	GLITCHPATCH	620	845	896	113
Qwen3.6-A3B	Original	580	815	855	98
	GLITCHPATCH	580	815	855	98
Mistral-7B	Original	500	770	775	80
	GLITCHPATCH	500	771	775	80
GLM-4.7	Original	550	800	830	95
	GLITCHPATCH	550	800	830	95
Deepseek-vl2	Original	520	785	815	88
	GLITCHPATCH	519	785	815	88

Table F2 shows the macro-averaged General Score computed from Table F1. The formula is:

$$\text{GeneralScore} = \frac{100}{4}\left(\frac{c_1}{1000} + \frac{c_2}{1000} + \frac{c_3}{1000} + \frac{c_4}{164}\right). \tag{8}$$

The ten-model Δ ranges from -0.025 to $+0.025$ pp, with a mean change of only $+0.008$ pp. Two models show minimal decreases (both caused by a single-item change). These small fluctuations indicate no systematic degradation of general benchmark performance.

Table F2. General Score and change (Δ in pp).

Model	Original	GLITCHPATCH	Δ (pp)
Llama-3.1-8B	66.48	66.51	+0.025
Llama-3.1-70B	79.87	79.87	+0.000
Gemma-4-31B	78.24	78.26	+0.025
Gemma-4-E4B	60.48	60.45	−0.025
Qwen3.8-27B	77.03	77.06	+0.025
Qwen3.6-27B	76.23	76.25	+0.025
Qwen3.6-35B-A3B	71.19	71.19	+0.000
Mistral-7B	63.32	63.35	+0.025
GLM-4.7	68.98	68.98	+0.000
Deepseek-vl2	66.41	66.39	−0.025
Mean	**70.82**	**70.83**	**+0.008**

F.2 FAILURE AND RETENTION CASES

Table F3 lists the possible failure and retention cases for GLITCHPATCH. Taking Qwen3.8-27B as an example, of 29,451 initial glitch tokens, 25,770 (87.50%) are successfully repaired and 1,473 (5.00%) are structurally unrepairable because no budget-feasible equivalent path exists in the vocabulary. The remaining 2,208 tokens (7.50%) have at least one feasible path but remain unrepaired for several reasons. First, approximately 0.1% of the initial targets have $m_+ = 0$ (all selection probes fail on the original representation), so no utility comparison is possible and the original encoding is retained (Algorithm 1, step 2a). Second, approximately 5.6% receive no stored rule because the utility of their best candidate does not exceed the default threshold ($0 < U(g, p) \leq \gamma$). Third, independent calibration rejected the selected candidate for approximately 0.1% of targets, confirming that the calibration stage provides a meaningful guard against overfitting to the selection probes. Fourth, approximately 1.7% of the initial targets receive a stored rule but do not pass the disjoint final GlitchQuiz evaluation, accounting for the difference between the default rule coverage (89.2%, Table D4) and the final FR (87.50%). On the subset with feasible paths and $m_+ > 0$, the final FR is 92.21%.

Table F3. Failure and retention cases for GLITCHPATCH.

Case	Observable evidence	System action
No budget-feasible path	Graph search: path count $= 0$	Retain original
All probes fail on original ($m_+ = 0$)	Selection probes: $m_+ = 0$	Retain original
Unreliable composition	Full-path probe failure	Reject candidate
Insufficient utility	$U(g, p) \leq \gamma$	Retain original
Calibration failure	Independent probe failure	Reject rule
Per-span limit	$\Delta(p) > B$ or content exceeds 64 bytes	Retain original
Unseen target	No key match	Pass through
Output-side failure	Input IDs unchanged	Run original model
Final-evaluation failure	Stored rule; final GlitchQuiz fails after substitution	Rule applied; counted as unfixed

G LANGUAGE-MODELING LOSS COMPARISON

Table G1 shows the relative token-level loss reduction for all four methods across ten models. The evaluation protocol is as follows: for each model, 1,000 successfully repaired glitch tokens are randomly sampled, and GPT-5.6-Terra generates a natural short passage (approximately 200 tokens) containing each target. The passages are fed to the target model using both original and repaired tokenizations, and the relative reduction in token-level average cross-entropy loss is computed. GLITCHPATCH achieves the largest loss reduction on all ten models (mean 13.47%), substantially exceeding GlitchCleaner (8.03%), GlitchEdit (5.46%), and GlitchProber (2.33%). The loss reduction of each method positively correlates with its FR: methods that repair more glitch tokens reduce loss on more text samples.

Table G1. Relative token-level loss reduction across methods (%).

Model	Prober	Edit	Cleaner	GLITCHPATCH
Llama-3.1-8B	2.2	6.4	8.3	**15.5**
Llama-3.1-70B	1.8	4.8	7.4	**11.9**
Gemma-4-31B	3.2	5.6	8.8	**12.5**
Gemma-4-E4B	1.5	5.8	6.4	**14.4**
Qwen3.8-27B	2.3	5.1	8.8	**13.9**
Qwen3.6-27B	2.7	5.4	8.5	**12.8**
Qwen3.6-A3B	1.8	4.9	7.0	**13.5**
Mistral-7B	2.6	5.4	8.1	**13.2**
GLM-4.7	3.8	7.0	11.2	**13.8**
Deepseek-vl2	1.4	4.2	5.8	**13.2**
Mean	2.33	5.46	8.03	**13.47**

H CONTEXTUAL REGRESSION AND DOWNSTREAM EVALUATION

This appendix evaluates two aspects not covered by the token-level FR/RR/GR metrics: whether local ID substitution degrades surrounding normal tokens in mixed inputs (Section H.1), and whether repair benefits generalize to downstream tasks beyond the three GlitchQuiz dimensions (Section H.2).

H.1 CONTEXTUAL REGRESSION ANALYSIS

Protocol. For each of three representative models (Mistral-7B, Llama-3.1-70B, Qwen3.8-27B), we randomly sample 500 glitch tokens that were successfully repaired by at least one method. We restrict to repaired tokens because only those receive altered input IDs; for unrepaired tokens the model input is identical, producing zero contextual change by construction. For each token, GPT-5.6-Terra generates a natural sentence (40–60 tokens) embedding the glitch-token string alongside distinctive normal-token content, together with a short-answer question whose answer depends solely on the adjacent normal tokens rather than the glitch token itself. We feed each sentence to the target model with and without each method's repair and compare exact-match accuracy on the short-answer questions. This isolates the effect of local ID substitution on surrounding context.

Table H1. Contextual regression: short-answer QA accuracy (%) on normal tokens adjacent to repaired glitch tokens. Each per-model accuracy is computed from 500 binary exact-match outcomes. Δ is the change in percentage points relative to the original (no-repair) baseline.

		GlitchProber		GlitchEdit		GlitchCleaner		GLITCHPATCH	
Model		Acc.	Δ	Acc.	Δ	Acc.	Δ	Acc.	Δ
Mistral-7B	(orig. 87.4)	86.8	−0.6	87.2	−0.2	87.0	−0.4	87.2	−0.2
Llama-3.1-70B	(orig. 91.6)	90.8	−0.8	91.2	−0.4	91.0	−0.6	91.8	+0.2
Qwen3.8-27B	(orig. 89.2)	88.4	−0.8	88.8	−0.4	88.8	−0.4	89.0	−0.2
Mean	(orig. 89.4)	88.7	−0.7	89.1	−0.3	88.9	−0.5	89.3	−0.1

Table H1 shows that GLITCHPATCH's accuracy changes are within ± 0.2 pp on every model (mean -0.1 pp), confirming that local ID substitution does not meaningfully degrade the processing of surrounding normal tokens. Among baselines, GlitchProber shows the largest per-token regression (mean -0.7 pp) because activation-layer intervention at the target position can propagate through attention to adjacent representations. GlitchEdit and GlitchCleaner show moderate changes as their embedding or adapter modifications alter the entire forward pass. GLITCHPATCH's advantage stems from its locality: only the input IDs at the target position change, and the self-attention mechanism naturally re-weights contributions from the replacement sub-tokens, which are individually well-learned by the model. As a supplementary check, we also evaluate on 500 uniformly sampled G_0

tokens; per-method mean Δ values remain within ± 0.1 pp of zero across all methods, confirming that no approach introduces measurable contextual harm at the population level.

H.2 Out-of-Distribution Downstream Evaluation

Tasks. We design three downstream tasks that test repair effectiveness beyond GlitchQuiz's repetition, spelling, and length dimensions: ❶ Entity QA: given a question about an entity whose name contains a glitch token, assess whether the model produces a factually relevant answer; scored by GPT-5.6-Terra on a binary relevance scale using a fixed judging prompt ("Is the response factually relevant to the named entity? Answer Yes or No."), with a 50-sample human validation subset showing 94% judge–human agreement; ❷ Verbatim Copy: the model is asked to repeat a full sentence (15–25 tokens) containing the glitch token exactly; scored by exact string match after whitespace normalization. Unlike GlitchQuiz's repetition task, which tests isolated single-token reproduction, verbatim copy embeds the target in a natural sentence context and requires faithful reproduction of the entire span; ❸ Code Identifier: a function stub uses a glitch-token string as a variable or function name; the model completes the function body; scored by whether the completion correctly references the identifier.

Sampling protocol. To avoid selection bias toward any single method, we uniformly sample 500 glitch tokens from the full detected set G_0 for each model, without conditioning on any method's repair success. Each method applies its repair where applicable; tokens that a given method does not repair retain their original tokenization. This ensures that all methods are compared on the same token set and that differences reflect both per-token repair quality and repair coverage.

Table H2. Out-of-distribution downstream task accuracy (%) on inputs containing glitch tokens. 500 glitch tokens uniformly sampled from G_0 per model. Best result per row in bold.

Task	Model	Orig.	Prober	Edit	Cleaner	Ours
Entity QA	Mistral-7B	41.8	43.2	49.6	56.4	**67.8**
	Llama-70B	48.2	50.8	57.0	63.8	**75.0**
	Qwen3.8-27B	44.6	46.6	52.8	59.2	**71.6**
Verb. Copy	Mistral-7B	38.2	39.8	45.4	52.8	**64.6**
	Llama-70B	43.6	45.6	52.0	59.6	**72.4**
	Qwen3.8-27B	40.8	42.4	48.2	55.0	**68.2**
Code Ident.	Mistral-7B	50.6	51.8	55.0	59.4	**66.8**
	Llama-70B	57.8	59.4	63.2	67.0	**74.0**
	Qwen3.8-27B	53.4	54.8	58.4	62.6	**70.0**
Overall mean		46.6	48.3	53.5	59.5	**70.0**

Results. Table H2 shows consistent improvements across all three tasks and models. GlitchPatch's overall mean accuracy is 70.0%, compared with 59.5% for GlitchCleaner, 53.5% for GlitchEdit, and 48.3% for GlitchProber. Entity QA and verbatim copy show the largest absolute gains (mean +26.6 and +27.5 pp over the original), confirming that repair restores the model's ability to recognize and reproduce previously opaque tokens; code-identifier completion also benefits substantially (mean +16.3 pp), testing a capability outside GlitchQuiz's three dimensions. The ranking across methods mirrors the FR ordering, with the gap driven by both per-token repair quality and repair coverage: on the sampled sets, GlitchPatch applies rules to 84–87% of the tokens while GlitchCleaner covers 70–74%, GlitchEdit 50–55%, and GlitchProber 35–40%. Because all methods are evaluated on the same uniformly sampled token set, these differences are not attributable to selection bias.