GlitchCleaner: Lightweight Glitch Tokens Repairing by Lossless Gated LoRA in Large Language Models

Yibo Fan, Jingru Li, Huan Li*

College of Artificial Intelligence, Nankai University
{yibofan, jingru_lee}@mail.nankai.edu.cn, lihuanss@nankai.edu.cn

Abstract

Large language models (LLMs) have been increasingly applied across a wide range of domains. However, recent studies have identified the presence of certain glitch tokens in their vocabularies, which can trigger hallucinations and lead to unpredictable or even harmful outputs. While various methods have been proposed to detect such tokens, effectively repairing them remains a key challenge for ensuring the reliability of LLMs. In this work, we propose GlitchCleaner, a lightweight yet effective approach to mitigate the adverse effects caused by glitch tokens. GlitchCleaner introduces auxiliary branches into specific components within selected layers of the model, enabling efficient and targeted token repair. These branches are implemented using the low-rank adaptation (LoRA) technique, adding less than 0.1% additional parameters to the original model. Furthermore, a gating mechanism dynamically controls the activation of these branches based on the model's input, ensuring precise intervention without disrupting normal inference behavior. Experimental results across multiple mainstream models demonstrate that our method achieves an average repair rate of 86.88%, representing an improvement of over 30% compared to existing approaches, while ensuring lossless preservation of the model's baseline capabilities and causing negligible impact on inference speed.

Code and Appendix —
https://github.com/FAVENO/GlitchCleaner

Introduction

Large Language Models (LLMs) are rapidly expanding in both popularity and scale. At the core of their language understanding lies the tokenizer and its associated vocabulary. These components decompose user queries into discrete tokens, which serve as the basic units for information processing and collectively form the input prompt (Zhang et al. 2024; Wang et al. 2025). Recent studies have identified a class of problematic tokens—referred to as "*glitch tokens*"—that can induce undesirable behaviors in LLMs (Rumbelow and mwatkins 2023; mwatkins and Rumbelow 2023a,b; mwatkins 2023). These behaviors include semantic distortion, output errors, offensive content generation, and vulnerability to prompt injection attacks (Geiping et al. 2024; Fell 2023). Such tokens, whether included intentionally or unintentionally, can lead to unpredictable outputs. This poses serious risks for downstream applications, especially in high-stakes domains such as finance (Nie et al. 2024; Wu et al. 2023; Yang, Liu, and Wang 2023), education (Jury et al. 2024; Wang et al. 2024), and software development (Hou et al. 2024; Nijkamp et al. 2023). As LLMs are increasingly integrated with external tools and agents for tasks like information retrieval and reasoning (Qin et al. 2024; Lewis et al. 2020), ensuring robustness against malformed or adversarial inputs has become a critical challenge.

Recent work (Land and Bartolo 2024; Li et al. 2024; Wu et al. 2024; Zhang et al. 2024) has primarily focused on efficiently identifying glitch tokens within model vocabularies, while research on effective remediation of these tokens remains limited. To the best of our knowledge, GlitchProber (Zhang et al. 2024) represents the only existing method for repairing glitch tokens, proposing an approach based on analyzing model activation patterns triggered by prompts containing glitch tokens. Specifically, Zhang et al. discovered that when processing inputs containing glitch tokens, certain model parameters exhibit significant activation changes compared to normal values. The method identifies glitch tokens through these differential activation patterns and attempts to repair them by correcting the abnormal activation values to normal levels. However, their repair success rate remains only 50.08%, still a long way from being usable.

Contribution

In this work, we propose GlitchCleaner, a lightweight and lossless method for repairing glitch tokens—inputs that trigger abnormal outputs in large language models. Our method offers the following advantages:

(1) We inject LoRA (Low-Rank Adaptation) branches only into the components that are primarily affected by glitch tokens. This lightweight design requires less than 0.1% of the original model's parameters, enabling efficient deployment with negligible resource overhead.

(2) The LoRA branches are gated and activate only in the presence of glitch tokens. For clean inputs, the model operates in its original form, ensuring that its baseline behavior remains unaffected. This gated design improves the reliability of GlitchCleaner across diverse inputs while preserving

*Corresponding author.

Type	Prompt	Response
Repetition	Question: Can you repeat the string 'Azalera' and return back to me? Answer: Here is the repeated string:	"saysay"
	Question: Can you repeat the string 'ennettuna' and return back to me? Answer: Here is the repeated string:	"abcabcabcabc"
Spelling	Question: Please can you spell out the string 'ispiele' with hyphens between each letter? Answer: Of course! The spelling of the string is:	e-x-i-m-p-l-e
	Question: Please can you spell out the string 'usztus' with hyphens between each letter? Answer: Of course! The spelling of the string is:	k-u-t-s-z-t-u-s
Length	Question: What is the length of this string 'Kapunoang'? Answer: The length of this string is:	3
	Question: What is the length of this string 'ennettuna'? Answer: The length of this string is:	2

Table 1: When presented with repetition, length, and spelling types of glitch tokens, the DeepSeek-V3-0324 model produces hallucinations and generates erroneous responses. Notably, DeepSeek-V3-0324 is one of the most recent and powerful open-source large language models, with 685 billion parameters.

the lossless nature of the repair process.

(3) Experimental results demonstrate that GlitchCleaner achieves an average repair rate of 86.88%, while maintaining superior performance in preserving the model's baseline capabilities and inference speed. Compared to GlitchProber (Zhang et al. 2024), it improves repair accuracy by more than 30% with no degradation in performance.

Background And Related Work

Glitch Tokens

Glitch tokens were initially discovered in GPT models (Rumbelow and mwatkins 2023; mwatkins 2023). When models encounter inputs containing these tokens, they may trigger anomalous behaviors, including irrelevant answers, semantic misinterpretations, response refusals, and offensive content generation (Fell 2023; mwatkins 2023; Li et al. 2024). Consequently, these tokens are commonly referred to as "*glitch tokens*". Table 1 presents examples of these three types of glitch tokens on the DeepSeek-V3-0324 model (DeepSeek-AI et al. 2025). Additional examples of glitch tokens are provided in Appendix A. Subsequent research has identified several distinctive characteristics of these tokens:

(1) They exhibit distinct features in embedding space compared to normal tokens, including reduced norms and clustering behaviors (Land and Bartolo 2024; Li et al. 2024; mwatkins and Rumbelow 2023a). Specifically, Fishing for Magikarp (Land and Bartolo 2024) identifies glitch tokens by analyzing token norm magnitudes in embedding space, indicating that these tokens are incompletely trained to some extent and share common characteristics in this regard. Meanwhile, glitchhunter (Li et al. 2024) proposed three detection tasks—repetition, length, and spelling—to identify glitch tokens and subsequently analyzed their distribution in embedding space, discovering that different types of glitch tokens typically cluster together. The use of multiple detection tasks demonstrates the diversity of glitch tokens.

(2) They generate abnormal activation values and prediction probabilities during model inference, manifesting as irregular attention patterns, anomalous MLP layer activations, and increased output uncertainty (Zhang et al. 2024; Wu et al. 2024). Specifically, GlitchProber (Zhang et al. 2024) employs support vector machines to learn the differences between intermediate activation values produced by normal and glitch tokens for detecting glitch tokens in models, achieving considerable detection accuracy. Meanwhile, GlitchMiner (Wu et al. 2024) discovered that glitch token inputs lead to increased uncertainty in model output predictions and leverages this observation to implement gradient-based detection of glitch tokens.

These approaches examine glitch tokens from different perspectives, providing important insights into the underlying causes and mechanisms of glitch token formation. While current research primarily focuses on leveraging these characteristics to identify glitch tokens, the development of effective repair mechanisms remains an equally critical challenge. This research area holds significant implications for the robustness and safety of large language models.

Gated Multi-Layer Perceptrons

Modern Transformer-based large language models widely employ Gated Multi-Layer Perceptrons (gMLPs) (Liu et al. 2021), which control information flow and processing through a gating mechanism. For input $X \in \mathbb{R}^{n \times d}$, where n represents the sequence length and d denotes the input dimension, the gMLP first projects the input to higher dimensions using two projection matrices W_1 and W_2:

$$Z_1 = XW_1 \qquad Z_2 = XW_2 \tag{1}$$

where $W_1, W_2 \in \mathbb{R}^{d \times d_m}$ are learnable projection matrices, and d_m represents the intermediate dimension. Subsequently, an activation function σ is applied element-wise to Z_1, resulting in $Z_1' = \sigma(Z_1)$. The gated output Z is then

computed through element-wise multiplication of Z_1' and Z_2:

$$Z = Z_1' \odot Z_2$$

Finally, the output is projected back to the original dimension using a learnable matrix W_3:

$$\text{output} = ZW_3$$

The MLP gate $\sigma(Z_1)$ and MLP data Z_2 work coordinately by processing attention information and integrating the relationships among tokens.

Low-Rank Adaptation (LoRA)

LoRA (Hu et al. 2022) is a widely adopted parameter-efficient fine-tuning technique that adapts pre-trained models by injecting trainable low-rank matrices into the existing weight structure. Instead of updating the full weight matrix, LoRA constrains the update to be low-rank, thereby significantly reducing the number of trainable parameters and memory overhead.

Formally, given a weight matrix $W \in \mathbb{R}^{d\times d}$ in a pre-trained model, LoRA freezes W and introduces a low-rank update $\Delta W = AB$, where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ are the learnable low-rank matrices, and $r \ll d$ denotes the rank. To maintain the scale of the original weights and avoid destabilizing the model, a scaling factor α is applied to the update, resulting in the final adapted weight:

$$W' = W + \Delta W = W + \frac{\alpha}{r}AB$$

This formulation allows the model to retain its pre-trained knowledge while adapting to new tasks with a minimal number of additional parameters.

GlitchCleaner: GlitchToken Repairing

In this work, we implement glitch token repair by applying LoRA adapters to the MLP layers of selected key transformer layers. By controlling the activation of the LoRA branches, we ensure that the model's original performance is preserved. This approach leverages LoRA's computational efficiency and low memory overhead while achieving effective repair performance.

Our method comprises four main steps: (1) filtering out tokens that are not meaningful to repair from the model's vocabulary; (2) identifying all glitch tokens by traversing the filtered model vocabulary; (3) integrating gated LoRA branches into the MLP layers of key model layers, which are activated only when the input prompt contains glitch tokens; and (4) constructing a dataset using these glitch tokens and their expected correct responses, then fine-tuning the model using the constructed dataset to train the gated LoRA branches.

Token Filter

Tokenizers divide input sentences into corresponding token IDs, which serve as inputs to the model. However, certain token IDs have special properties that make them unsuitable for creating appropriate prompts to verify whether they are glitch tokens, and treating them as glitch tokens would be meaningless. Following established protocols (Land and Bartolo 2024; Wu et al. 2024), we first filter out special tokens and non-encodable tokens from the model vocabulary. This process involves decoding each token, re-encoding it, and then performing classification. We exclude the following categories of tokens from the classification results:

(1) SPECIAL tokens: tokens with special semantic meaning in the model, such as '<s>', '<unk>', and '<endoftext>'. These are manually added control markers; for example, '<endoftext>' typically indicates the model's inference end marker. (2) UNDECODEABLE tokens: tokens that cannot be decoded, typically containing invalid characters. These are usually part of UTF-8 encoding and cannot be converted to Unicode characters (Land and Bartolo 2024). (3) UNREACHABLE tokens: these are tokens whose decoded string, when re-encoded, yields a different token ID than the original. In other words, the original token IDs cannot be reproduced through the model's tokenization process. This phenomenon is often attributed to the specific rules governing the tokenization process, such as a longest-match-first strategy. As a result, such tokens will never appear during the encoding of any input sentence, making their detection unnecessary.

This filtering step prevents these tokens from being incorrectly classified as glitch tokens due to encoding-decoding issues. In the Llama-2-7B-chat model (Touvron et al. 2023), we filter 229 tokens out of 32,000 tokens, including 3 special tokens, 2 undecodeable tokens, and 224 unreachable tokens. The results for other models are provided in Appendix B.

Glitch Token Identification

We then identify glitch tokens by systematically traversing the remaining vocabulary using a repeated token input task, which is widely employed in prior research (Zhang et al. 2024; Wu et al. 2024; Li et al. 2024). We employ the following prompt template and evaluate the model's ability to produce the correct response:

"Can you repeat the string '{token}' and return it back to me? Answer: Here is the repeated string:"

To obtain definitive results, we include positive directive expressions in the prompt, such as "Answer: Here is the repeated string:" (Li et al. 2024), which guides the model to actively complete the task and improves detection accuracy. Additionally, to ensure consistency in model outputs, we set the temperature parameter to 0, which ensures that the model produces identical outputs for the same input. Under these settings, if the model produces unexpected outputs, we classify the corresponding token as a glitch token.

We traverse the 31,771 filtered tokens in the Llama-2-7B-chat model and identified 4,743 glitch tokens. The number of glitch tokens for other models can be found in the Results section.

Gated LoRA Architecture

Previous work by GlitchProber (Zhang et al. 2024) demonstrated that glitch tokens induce abnormal activations in both the gating and data components of gMLP layers within key model layers. We further analyze the differences in activation values induced by glitch tokens and normal tokens

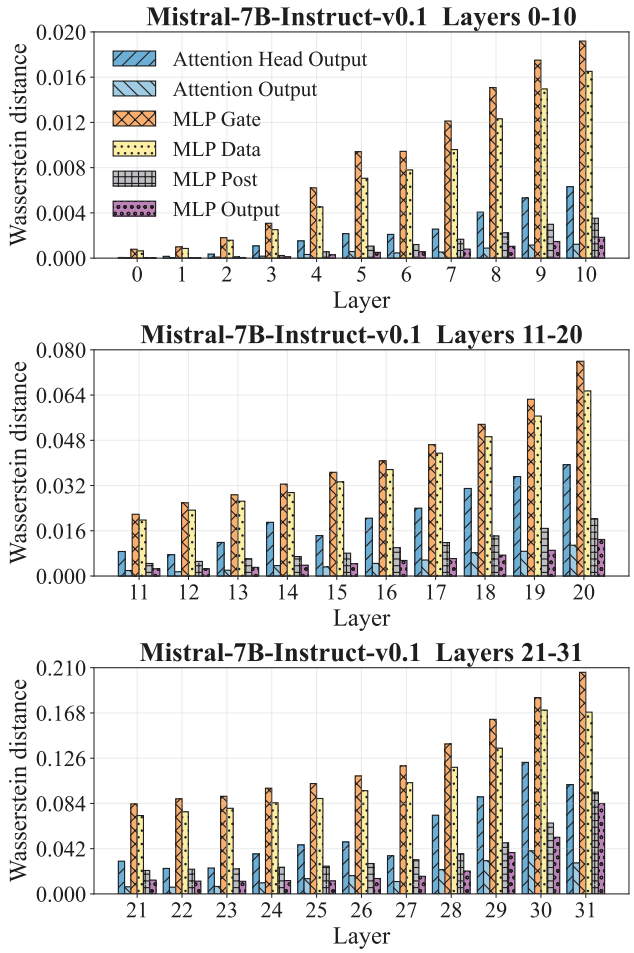

Figure 1: Wasserstein distance between the activations of glitch tokens and normal tokens across different components within Transformer blocks in the Mistral-7B-Instruct-v0.1 model. The results illustrate that the differences become more pronounced in deeper layers, particularly in the MLP Gate and MLP Data components.

across various components within Transformer blocks. In the Mistral-7B-Instruct-v0.1 model, we select the previously mentioned glitch tokens and an equal number of normal tokens, construct prompts using these tokens, and examine the corresponding model activations. To quantify the magnitude of the difference in activations between normal and glitch tokens, we compute the Wasserstein distance (Vaserstein 1969) between the activation outputs elicited by the two token types across different components within each Transformer block. The detailed computation procedure is provided in Appendix C. To compare variations across different layers, we analyze all layers of the model. We conduct the same experiments on the other models to verify the generality of this phenomenon. Results for additional models can be found in Appendix D.

Experimental results in Figure 1 reveal that the differences in activation values between glitch tokens and normal tokens become significantly larger after passing through the linear layers in the MLP gate and MLP data components, compared to the differences observed in the attention-related components such as the Attention Head Output and Attention Output. These observations suggest that MLP gate and MLP data play a key role in amplifying the differences between glitch and normal tokens, potentially due to parameter configurations that cause the activations of glitch tokens to deviate from those of normal tokens. Notably, both MLP gate and MLP data are linear transformations applied in parallel to the attention output, rather than sequential layers. Moreover, by comparing across layers, we observe that the discrepancy between glitch and normal tokens becomes increasingly pronounced in deeper layers of the model.

Based on the above observations, we add *gated LoRA branches* only to the MLP gate and MLP data components within the layers near the model's output section, which we refer to as "*key layers*". This selection is based on the following considerations: (1) The difference between normal tokens and glitch tokens becomes more apparent in the model's later layers, making parameter training in these layers more effective for learning the distinctions between the two token types. (2) The parameters in the MLP gate and MLP data layers contribute disproportionately to the activation discrepancies associated with glitch tokens. This may be an important contributing factor to the abnormal outputs produced by such tokens. (3) MLP and attention mechanisms serve different functions within the model (Zhang et al. 2024). MLP layers reflect the model's understanding and processing of individual token information, while attention patterns primarily capture relationships between tokens. Modifying attention values may affect the model's overall reasoning accuracy and introduce noise. By adjusting only the MLP layer activation values, we can minimize impact on the model's inference process while maintaining targeted correction capabilities.

Additionally, the gated LoRA branches are activated only when the model detects glitch tokens in the input, which further ensures that normal inference remains unaffected. The LoRA branches are trained using glitch tokens paired with their corresponding correct outputs, enabling automatic adjustment of MLP layer values and glitch token correction. Specifically, we integrate LoRA branches into the projection matrices W_1 and W_2 from Equations (1) :

$$W' = W + \lambda \cdot \frac{\alpha}{r} AB$$

When the model input contains glitch tokens, λ is set to 1; otherwise, λ is set to 0. The detailed workflow is illustrated in Figure 2.

Gated LoRA Branches Parameter Fine-tuning

We construct a question-answering dataset using the identified glitch tokens for subsequent fine-tuning. The dataset includes the same prompt templates described in Glitch Token Identification, paired with their corresponding expected responses. Additionally, to validate the model's ability to repair various types of glitch tokens, we select additional categories of glitch tokens in subsequent experiments and con-

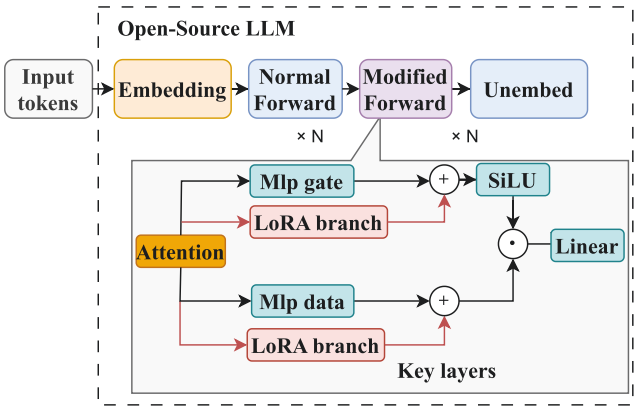

Figure 2: Workflow of GlitchCleaner: The red arrows indicate the activation of LoRA branches for repair only when glitch tokens are detected in the input.

struct corresponding datasets. Details are provided in Results section.

We utilize the dataset to fine-tune the LoRA branches, enabling the large model to repair these glitch tokens. During fine-tuning, consistent with common methodologies, we froze the parameters of the main model body and exclusively train the parameters of the LoRA branches. Due to the relatively small number of parameters involved, this fine-tuning approach requires limited computational cost. The pseudocode for the algorithm of GlitchCleaner is shown in Algorithm 1.

Algorithm 1: GlitchCleaner

Input: Model M; Set of key layers K; Set of glitch tokens G; Input prompt P
Parameter: LoRA branches parameters: rank r and scalar α
Output: Model output O

#Stage 1: Add LoRA branches to key layers
for each layer l in M.layers **do**
  if $l \in K$ **then**
    l.MLP.Gate $\leftarrow l$.MLP.Gate $+ \lambda \times$ LoRA(r, α)
    l.MLP.Data $\leftarrow l$.MLP.Data $+ \lambda \times$ LoRA(r, α)
  end if
end for
#Stage 2: Fine-tune LoRA parameters
$D \leftarrow$ ConstructDataset(G)
Set $\lambda \leftarrow 1$
$M' \leftarrow$ FineTune(M, D, r, α)
#Stage 3: Inference with repaired model
$I \leftarrow$ Tokenize(P)
if $I \cap G \neq \emptyset$ **then**
  $\lambda \leftarrow 1$
else
  $\lambda \leftarrow 0$
end if
$O \leftarrow M'$.generate(I)
return O

Differences From GlitchProber

To the best of our knowledge, GlitchProber (Zhang et al. 2024) is the only prior work that explicitly addresses the repair of glitch token effects. Both GlitchProber and GlitchCleaner focus on the influence of MLP components in layers near the output on glitch token generation; however, they differ in their approaches to repairing anomalous values in these components. GlitchProber selects a small subset of neurons in the MLP as key neurons, using a screening method that identifies neurons that are either activated in most cases (99%) or remain inactive in most cases (99%) for normal tokens. Using these as references, it compares the values of these key neurons in glitch tokens with their corresponding values in normal tokens, attempting to repair any differences by adjusting them to the activation levels of normal tokens. In contrast, GlitchCleaner does not differentiate between MLP layer neurons but relies on training LoRA branches parameters to determine how to repair individual neurons.

Our approach offers two advantages: first, it avoids the risk of inappropriately set thresholds affecting normal neurons; second, since different types of glitch tokens may require different activation value repairs, handling these variations correctly may demand more sophisticated processing. Training LoRA parameters with various types of glitch tokens eliminates the need for additional processing steps. Regarding model compatibility, LoRA technology, as a mature technique, can be conveniently integrated into models, making GlitchCleaner more favorable for inference speed and hardware optimization. Details are provided in Result section.

Results

Experimental Setup

We set the LoRA rank parameter r to 4 and the scaling factor α to 4. This minimal rank value proves sufficient to achieve effective repair performance. For the key layers, we adopt the configuration from GlitchProber, primarily targeting the posterior layers of the model. For example, in Llama-2-7b-chat, we select layers 19 through 28 as the key layers.

As the r we set is very low and replacements were only made in the MLP blocks of the key layers, the number of parameters added by the LoRA branches is less than 0.1% of the original model parameters.

Repair Rate

We fine-tune the model with added LoRA branches using a dataset built from glitch tokens filtered by the model, then evaluate whether the repaired model could generate correct responses.

To thoroughly evaluate the practical effectiveness of our approach, we follow GlitchProber (Zhang et al. 2024) and conduct evaluations on the same set of models to enable direct comparison. Specifically, we evaluate six widely used models, including Llama-2-7b-chat (Touvron et al. 2023), Mistral-7B-Instruct-v0.1 (Jiang et al. 2023), Qwen-7B-Chat (Bai et al. 2023), Gemma-2b-it (Team et al. 2024)

Model	Glitchtokens	Metric	Method	
			GlitchProber	GlitchCleaner
Llama-2-7b-chat	4743	Repaired Tokens Repair Rate	2968 62.58%	**4210** **88.76%**
Gemma-2b-it	29831	Repaired Tokens Repair Rate	14548 48.77%	**20697** **69.38%**
Mistral-7B-Instruct-v0.1	2539	Repaired Tokens Repair Rate	956 37.65%	**2407** **94.80%**
Qwen-7B-Chat	27686	Repaired Tokens Repair Rate	13320 48.11%	**24582** **88.79%**
Yi-6B-Chat	5985	Repaired Tokens Repair Rate	3188 53.27%	**5547** **92.68%**
Average		Repair Rate	50.08%	**86.88%**

Table 2: Comparison of repair rates between GlitchCleaner and GlitchProber on repeat glitch type. GlitchCleaner demonstrates superior repair rates across all tested models.

Glitch tokens type	Spelling	Length
Repair rate	94.43%	92.45%

Table 3: The repair effectiveness of GlitchCleaner on glitch tokens detected using the Spelling and Length detection methods.

and Yi-6B-Chat (AI et al. 2025), where the GlitchProber results are taken from their original paper. We calculate the accuracy rate as the ratio of correct responses to the total number of faulty tokens and compare our method with GlitchProber's repair approach. The results are presented in Table 2.

The results show that our method, GlitchCleaner, achieves better repair rates across these models. This indicates that training the LoRA branches parameters effectively helps the model correct wrong outputs when processing glitch tokens.

Additionally, to test the effectiveness of our method in repairing various types of errors, we also create two other question-answering datasets using these glitch tokens: one focused on spelling tasks and another on token length output tasks.

To ensure proper answer formatting and accurate glitch token filtering, we include specific few-shot examples in the prompts. The specific details are presented in Appendix G. Then we use GlitchCleaner to repair these tokens and measure the repair rates on the LLaMA-2 model. As shown in Table 3, GlitchCleaner achieves good repair performance across different types of glitch tokens, demonstrating the broad applicability of this approach.

Unaffected Performance

To evaluate how the gated branches affects model performance, we follow GlitchProber and evaluate the modified model on two widely used benchmark datasets, GSM8K (Cobbe et al. 2021) and MMLU (Hendrycks et al. 2021), using the lm-evaluation-harness (Gao et al. 2024). Then compare our method with GlitchProber and full fine-tuning approaches. The full fine-tuning method involves training the model on a dataset built from the selected glitch tokens with correct answers. The results are shown in Table 4.

The results show that GlitchCleaner does not affect model performance and maintains the original model's capabilities. In contrast, direct fine-tuning on the original model leads to performance degradation.

Extended Evaluation

We further test our method on more advanced models (Llama-3.1-8B-Instruct (Grattafiori et al. 2024), Qwen3-8B (Yang et al. 2025), Mistral-7B-Instruct-v0.3 (Jiang et al. 2023) and additional benchmark datasets, including C-Eval (Huang et al. 2023), MetaBench (Kipnis et al. 2025), PIQA(Bisk et al. 2019), and AGIEval (Zhong et al. 2023). The results, reported in Table 5, are consistent with our previous findings.

We also observe that the occurrence of glitch tokens is reduced in these more advanced models. For example, Mistral-7B-Instruct-v0.1 contains 2,539 glitch tokens, whereas Mistral-7B-Instruct-v0.3 contains only 1,020, and the repair effect is correspondingly improved. This trend arises because glitch tokens are essentially untrained tokens (Land and Bartolo 2024), and their occurrence probability decreases as the scale and coverage of training data increase. This also indirectly explains why fine-tuning on a dataset constructed from glitch tokens can effectively repair the model.

Model Inference Speed Comparison

We evaluate the inference speed of the Llama-2-7b-chat model after GlitchCleaner correction on an H200 GPU and compare it with the original model's inference speed and

Model	Task	Score			
		Original model	GlitchCleaner	GlitchProber	Finetuning
Llama-2-7b-chat	GSM8K	23.05	**23.88**	21.23	21.54
	MMLU	46.38	46.23	**46.31**	42.34
Gemma-2b-it	GSM8K	10.99	9.33	**10.77**	2.67
	MMLU	38.14	**40.27**	38.16	33.62
Mistral-7B-Instruct-v0.1	GSM8K	34.57	**34.04**	32.98	25.54
	MMLU	53.45	**53.38**	53.37	53.38
Qwen-7B-Chat	GSM8K	47.84	47.46	**48.60**	40.33
	MMLU	54.27	53.82	**54.25**	54.14
Yi-6B-Chat	GSM8K	40.71	**40.49**	11.22	37.22
	MMLU	61.63	61.58	52.55	**61.67**
Average	GSM8K	31.43	**31.04**	24.96	25.46
	MMLU	50.77	**51.06**	48.93	49.03

Table 4: Performance comparison of the original model, GlitchCleaner-corrected model, GlitchProber-corrected model, and full fine-tuned model on GSM8K and MMLU datasets.

Model	Repair Rate	Performance Score					
		GSM8K	MMLU	C-Eval	MetaBench	PIQA	AGIEval
Llama-3.1-8B-Instruct	95.09%	75.58	68.05	53.86	55.69	79.86	42.40
+ GlitchCleaner		77.33	68.29	54.08	55.06	79.49	42.44
Qwen3-8B	98.44%	88.40	72.97	79.63	57.23	76.82	56.80
+ GlitchCleaner		87.83	72.65	79.32	57.04	75.51	57.88
Mistral-7B-Instruct-v0.3	93.73%	50.11	59.73	44.42	61.57	81.55	36.62
+ GlitchCleaner		48.75	59.89	43.61	62.00	81.66	36.80

Table 5: Repair rate on various 7–8B SOTA models and performance comparison across multiple benchmarks before and after applying GlitchCleaner.

Model	Inference speed (tokens/sec)
Original model	66.30
GlitchCleaner	62.83
GlitchProber	11.82

Table 6: Inference speeds of the original model, GlitchCleaner, and GlitchProber on H200 GPU.

that of a simple GlitchProber implementation. The results are presented in Table 6. The results indicate that GlitchCleaner has limited impact on the model's inference speed.

Ablation Study

We further conduct an ablation study to better understand the contributions of different components in our design. Due to space constraints, we present additional experimental results in the appendix. Specifically, Appendix E reports experiments involving the insertion of gated LoRA into other components of the Transformer blocks, and Appendix F analyzes the effects of varying the r and α hyperparameters.

Conclusion

In this paper, we present GlitchCleaner, a method that adds LoRA branches to specific parts of large language models and trains these branches using glitch tokens paired with their correct responses to achieve repair effects. We first identify glitch tokens requiring remediation, then create a dataset to train the LoRA branches for automatic correction of output errors associated with these tokens. Our approach demonstrates high repair rates across various models and token error types while maintaining model performance, utilizing less than 0.1% of the original model parameters with highly efficient training time and space requirements. This shows its potential for enhancing large language model reliability and robustness. Future work could explore improved dataset construction methods, more accurate glitch token detection techniques, and the adaptation of this approach across different model architectures.

Acknowledgments

H. Li was supported by the NSF China (No. 62476142).

References

AI, .; :; Young, A.; Chen, B.; Li, C.; Huang, C.; Zhang, G.; Zhang, G.; Wang, G.; Li, H.; Zhu, J.; Chen, J.; Chang, J.; Yu, K.; Liu, P.; Liu, Q.; Yue, S.; Yang, S.; Yang, S.; Xie, W.; Huang, W.; Hu, X.; Ren, X.; Niu, X.; Nie, P.; Li, Y.; Xu, Y.; Liu, Y.; Wang, Y.; Cai, Y.; Gu, Z.; Liu, Z.; and Dai, Z. 2025. Yi: Open Foundation Models by 01.AI. arXiv:2403.04652.

Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; Hui, B.; Ji, L.; Li, M.; Lin, J.; Lin, R.; Liu, D.; Liu, G.; Lu, C.; Lu, K.; Ma, J.; Men, R.; Ren, X.; Ren, X.; Tan, C.; Tan, S.; Tu, J.; Wang, P.; Wang, S.; Wang, W.; Wu, S.; Xu, B.; Xu, J.; Yang, A.; Yang, H.; Yang, J.; Yang, S.; Yao, Y.; Yu, B.; Yuan, H.; Yuan, Z.; Zhang, J.; Zhang, X.; Zhang, Y.; Zhang, Z.; Zhou, C.; Zhou, J.; Zhou, X.; and Zhu, T. 2023. Qwen Technical Report. arXiv:2309.16609.

Bisk, Y.; Zellers, R.; Bras, R. L.; Gao, J.; and Choi, Y. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language. arXiv:1911.11641.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.

DeepSeek-AI; Liu, A.; Feng, B.; et al. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437.

Fell, M. 2023. A Search for More ChatGPT / GPT-3.5 / GPT-4 "Unspeakable" Glitch Tokens. Blog Post.

Gao, L.; Tow, J.; Abbasi, B.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; Le Noac'h, A.; Li, H.; McDonell, K.; Muennighoff, N.; Ociepa, C.; Phang, J.; Reynolds, L.; Schoelkopf, H.; Skowron, A.; Sutawika, L.; Tang, E.; Thite, A.; Wang, B.; Wang, K.; and Zou, A. 2024. The Language Model Evaluation Harness.

Geiping, J.; Stein, A.; Shu, M.; Saifullah, K.; Wen, Y.; and Goldstein, T. 2024. Coercing LLMs to do and reveal (almost) anything. *CoRR*, abs/2402.14020.

Grattafiori, A.; Dubey, A.; Jauhri, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; and Steinhardt, J. 2021. Measuring Massive Multitask Language Understanding. arXiv:2009.03300.

Hou, X.; Zhao, Y.; Liu, Y.; Yang, Z.; Wang, K.; Li, L.; Luo, X.; Lo, D.; Grundy, J.; and Wang, H. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations, ICLR 2022*.

Huang, Y.; Bai, Y.; Zhu, Z.; Zhang, J.; Zhang, J.; Su, T.; Liu, J.; Lv, C.; Zhang, Y.; Lei, J.; Fu, Y.; Sun, M.; and He, J. 2023. C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models. arXiv:2305.08322.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Scao, T. L.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2023. Mistral 7B. arXiv:2310.06825.

Jury, B.; Lorusso, A.; Leinonen, J.; Denny, P.; and Luxton-Reilly, A. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Australasian Computing Education Conference, ACE 2024*.

Kipnis, A.; Voudouris, K.; Buschoff, L. M. S.; and Schulz, E. 2025. metabench – A Sparse Benchmark of Reasoning and Knowledge in Large Language Models. arXiv:2407.12844.

Land, S.; and Bartolo, M. 2024. Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2024*.

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.; Rocktäschel, T.; Riedel, S.; and Kiela, D. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Conference on Neural Information Processing Systems , NeurIPS 2020*.

Li, Y.; Liu, Y.; Deng, G.; Zhang, Y.; Song, W.; Shi, L.; Wang, K.; Li, Y.; Liu, Y.; and Wang, H. 2024. Glitch Tokens in Large Language Models: Categorization Taxonomy and Effective Detection. *Proc. ACM Softw. Eng.*, 1(FSE): 2075–2097.

Liu, H.; Dai, Z.; So, D. R.; and Le, Q. V. 2021. Pay Attention to MLPs. In *Conference on Neural Information Processing Systems, NeurIPS 2021*.

mwatkins. 2023. The petertodd phenomenon. Blog Post.

mwatkins; and Rumbelow, J. 2023a. SolidGoldMagikarp II: technical details and more recent findings. Blog Post.

mwatkins; and Rumbelow, J. 2023b. SolidGoldMagikarp III: Glitch token archaeology. Blog Post.

Nie, Y.; Kong, Y.; Dong, X.; Mulvey, J. M.; Poor, H. V.; Wen, Q.; and Zohren, S. 2024. A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges. arXiv:2406.11903.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv:2203.13474.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; Zhao, S.; Hong, L.; Tian, R.; Xie, R.; Zhou, J.; Gerstein, M.; Li, D.; Liu, Z.; and Sun, M. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations, ICLR 2024*.

Rumbelow, J.; and mwatkins. 2023. SolidGoldMagikarp(plus, prompt generation). Blog Post.

Team, G.; Mesnard, T.; Hardin, C.; et al. 2024. Gemma: Open Models Based on Gemini Research and Technology. arXiv:2403.08295.

Touvron, H.; Martin, L.; Stone, K.; et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Vaserstein, L. N. 1969. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3): 64–72.

Wang, D.; Li, Y.; Jiang, J.; Ding, Z.; Luo, Z.; Jiang, G.; Liang, J.; and Yang, D. 2025. Tokenization Matters! Degrading Large Language Models through Challenging Their Tokenization. arXiv:2405.17067.

Wang, S.; Xu, T.; Li, H.; Zhang, C.; Liang, J.; Tang, J.; Yu, P. S.; and Wen, Q. 2024. Large Language Models for Education: A Survey and Outlook. *CoRR*, abs/2403.18105.

Wu, S.; Irsoy, O.; Lu, S.; Dabravolski, V.; Dredze, M.; Gehrmann, S.; Kambadur, P.; Rosenberg, D.; and Mann, G. 2023. BloombergGPT: A Large Language Model for Finance. arXiv:2303.17564.

Wu, Z.; Gao, H.; Wang, P.; Zhang, S.; Liu, Z.; and Lian, S. 2024. GlitchMiner: Mining Glitch Tokens in Large Language Models via Gradient-based Discrete Optimization. arXiv:2410.15052.

Yang, A.; Li, A.; Yang, B.; et al. 2025. Qwen3 Technical Report. arXiv:2505.09388.

Yang, H.; Liu, X.-Y.; and Wang, C. D. 2023. FinGPT: Open-Source Financial Large Language Models. arXiv:2306.06031.

Zhang, Z.; Bai, W.; Li, Y.; Meng, M. H.; Wang, K.; Shi, L.; Li, L.; Wang, J.; and Wang, H. 2024. GlitchProber: Advancing Effective Detection and Mitigation of Glitch Tokens in Large Language Models. In *IEEE/ACM International Conference on Automated Software Engineering*, ASE 24.

Zhong, W.; Cui, R.; Guo, Y.; Liang, Y.; Lu, S.; Wang, Y.; Saied, A.; Chen, W.; and Duan, N. 2023. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. arXiv:2304.06364.