GlitchMiner: Mining Glitch Tokens in Large Language Models via Gradient-based Discrete Optimization

Zihui Wu[1], **Haichang Gao**[1*], **Ping Wang**[1], **Shudong Zhang**[1], **Zhaoxiang Liu**[2, 3], **Shiguo Lian**[2, 3†]

[1]School of Computer Science and Technology, Xidian University
[2]Data Science & Artificial Intelligence Research Institute, China Unicom
[3]Unicom Data Intelligence, China Unicom
zihui@stu.xidian.edu.cn, hchgao@xidian.edu.cn, liansg@chinaunicom.cn

Abstract

Glitch tokens—inputs that trigger unpredictable or anomalous behavior in Large Language Models (LLMs)—pose significant challenges to model reliability and safety. Existing detection methods primarily rely on heuristic embedding patterns or statistical anomalies within internal representations, limiting their generalizability across different model architectures and potentially missing anomalies that deviate from observed patterns. We introduce **GlitchMiner**, an behavior-driven framework designed to identify glitch tokens by maximizing predictive entropy. Leveraging a gradient-guided local search strategy, GlitchMiner efficiently explores the discrete token space without relying on model-specific heuristics or large-batch sampling. Extensive experiments across ten LLMs from five major model families demonstrate that GlitchMiner consistently outperforms existing approaches in detection accuracy and query efficiency, providing a generalizable and scalable solution for effective glitch token discovery.

Code — https://github.com/wooozihui/GlitchMiner

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities and are increasingly deployed in high-stakes domains such as code generation (Jiang et al. 2024; Chen et al. 2021; Nijkamp et al. 2022), healthcare (Goel et al. 2023; Wang, Ma, and Chen 2023), and education (Wang et al. 2024; Jury et al. 2024). As reliance on LLMs grows, ensuring their safety and output reliability becomes imperative.

A particularly concerning threat arises from a class of anomalous inputs known as *glitch tokens* (LessWrong Community 2023). These tokens can induce LLMs to generate unexpected, erratic, or even policy-violating outputs (Geiping et al. 2024), undermining trust in their behavior. For example, as illustrated in Figure 1, when prompted to repeat the word "Mediabestanden," Llama2-7b-chat outputs "hello world"—a semantically unrelated response. This phenomenon indicates that certain tokens can trigger unpredictable model behavior, even under simple and deterministic instructions.

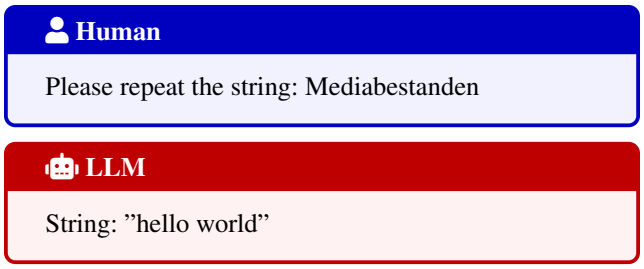

Figure 1: An illustrative example of how glitch token causing Llama2-7b-chat to fail a simple repetition task. More examples can be found in Appendix B.

In response, several studies have been proposed to detect glitch tokens (Land and Bartolo 2024a; Li et al. 2024; Zhang et al. 2024). These methods typically rely on heuristic observations or statistical patterns in token embeddings or latent representations, such as small embedding norms or localized embedding clusters. Although effective in certain scenarios, such heuristics have two main limitations. First, the reliance on specific observed statistical features may not generalize well across different architectures or training setups. Second, tokens that do not conform to these empirically observed patterns may remain undetected, potentially limiting the overall coverage and robustness of these detection methods.

To address these limitations, we propose a behavior-driven perspective. Rather than searching for fixed patterns within the model, we hypothesize that glitch tokens are outliers in the model's learned distribution, and thus elicit *uncertain* predictions. This uncertainty can be quantified via the entropy of the model's next-token distribution—high entropy indicates that the model is indecisive, providing a proxy for behavioral instability.

Building on this insight, we present **GlitchMiner**, a novel framework that identifies glitch tokens by maximizing output entropy. The core of GlitchMiner is a *gradient-guided local search*, which efficiently navigates the discrete token space to identify high-entropy candidates. Unlike prior gradient-based discrete optimization strategies that suffer from inaccurate approximations or require large batch sampling (Shin et al. 2020; Zou et al. 2023), our method restricts updates to neighboring tokens in the embedding space. This ensures

*Corresponding author.
†Corresponding author.

accurate Taylor-based entropy estimates and improves search efficiency.

Our Contributions are summarized as follows:

- We propose a behavior-driven framework for glitch token detection, leveraging output entropy as an architecture-agnostic measure of model uncertainty to ensure broad applicability across diverse models.
- We introduce a gradient-guided local search algorithm that accurately estimates entropy gradients within the discrete token space, eliminating the dependence on large-batch sampling and improving search efficiency.
- Extensive experiments across 10 LLMs from 5 major model families demonstrate that GlitchMiner consistently surpasses state-of-the-art methods in detection accuracy, establishing it as a robust and effective solution for glitch token detection.

The remainder of the paper is organized as follows: Section 2 reviews related work; Section 3 describes the GlitchMiner methodology; Section 4 presents experiments and ablations; and Section 5 concludes with future directions.

2 Background and Related Work

2.1 Glitch Token Definition

In practice, the identification of glitch tokens relies on specific behavioral tests. The most common approach, used in prior work (Land and Bartolo 2024a; Li et al. 2024; Zhang et al. 2024), is the **repetition task**. In such a task, a model is prompted to repeat a given token t; a failure to accurately reproduce the token is taken as evidence of it being a glitch. This single-test method, however, is fragile. The outcome can be highly sensitive to the specific wording of the prompt, leading to inconsistent results and potential false positives.

To establish a more robust and replicable definition, we move beyond single-prompt evaluations. We classify a token t as a *glitch token* only if it fails the repetition task consistently across a diverse set of m prompt templates. This stricter criterion is formalized as follows: given a set of m templates, $h_i(\cdot)\}_{i=1}^{m}$, the failure rate for a token t is defined as:

$$\text{fail}(t) := \frac{1}{m} \sum_{i=1}^{m} \mathbb{1} \left[\arg\max_{v \in \mathcal{V}} P(v \mid h_i(t)) \neq t \right] \quad (1)$$

A token is then identified as a glitch if and only if its failure rate is absolute, i.e., $\text{fail}(t) = 1$. This rigorous cross-verification process effectively filters out template-specific artifacts, ensuring that our analysis focuses exclusively on consistently unstable tokens.

2.2 Glitch Token Detection

A series of methods have been proposed to *efficiently rank or localise* glitch tokens in large vocabularies:

Magikarp (Land and Bartolo 2024a) adopts a lightweight heuristic, screening tokens with atypically small ℓ_2 embedding norms before verifying them via the repetition task.

GlitchHunter (Li et al. 2024) observes that glitch tokens tend to cluster in embedding space. It builds a token-embedding graph and applies Leiden clustering (Traag, Waltman, and Van Eck 2019), followed by iterative hypothesis testing to refine each cluster.

GlitchProber (Zhang et al. 2024) shifts the focus from embeddings to *internal activations*. It projects hidden states and attention outputs with PCA, then trains SVM classifiers to flag activation outliers; a mitigation step masks offending neurons at inference time.

While effective, these approaches share a common limitation: they rely heavily on heuristic observations of embedding or activation patterns. Such reliance may reduce their generalizability across different architectures, and tokens not exhibiting these typical patterns may remain undetected.

In contrast, GlitchMiner employs gradient-guided discrete optimization to maximize output entropy, an architecture-agnostic indicator of predictive uncertainty that facilitates broad generalization across diverse LLMs.

2.3 Gradient-based Discrete Optimization

Gradient-based discrete optimization methods (Ebrahimi et al. 2017; Shin et al. 2020; Zou et al. 2023; Wen et al. 2024) leverage gradient information to predict how individual tokens impact the loss function. These approaches typically treat the one-hot encoding of tokens or token embeddings as continuous vectors to compute gradients, guiding token replacements for optimization.

HotFlip (Ebrahimi et al. 2017) uses the one-hot encoding of the tokens to calculate the gradients and selects the token with the largest negative gradient to replace the current token, with the goal of minimizing the loss. However, it only evaluates one candidate token per iteration, which can lead to suboptimal predictions and reduced accuracy.

AutoPrompt (Shin et al. 2020) improves upon HotFlip by evaluating multiple candidate tokens in each iteration. Instead of relying on gradients from one-hot encodings, it utilizes token embedding gradients for loss estimation, enhancing prediction accuracy by considering a broader range of potential token replacements.

GCG (Zou et al. 2023) extends HotFlip by incorporating multi-candidate token selection, similar to AutoPrompt, but it still uses the one-hot encoding of tokens to compute gradients for loss estimation. Notably, GCG has been applied to automated *jailbreaks* (Shen et al. 2023) in LLMs, efficiently searching for adversarial suffixes.

AutoPrompt and GCG both rely on **large batch sampling** to mitigate inaccuracies in gradient-based prediction. We identified that these inaccuracies arise from the inaccuracy of Taylor expansions when input tokens are distant from the original points. This overlooks a fundamental condition of Taylor approximation: its accuracy is highest for points close to the reference point.

Building on these works, we introduce a **local search strategy** in our approach. This improvement enables us to achieve high precision in gradient estimation without relying on large batch sampling, by focusing on a smaller, localized token space. By addressing the core issue of Taylor approximation accuracy, our method allows for more efficient and accurate

Algorithm 1: The GlitchMiner Pipeline

Input: LLM model $f(\cdot)$, Full vocabulary $\mathcal{T}$, Max iterations N
Output: Verified glitch token set $\mathcal{G}$
Initialization
$\mathcal{T}^* \leftarrow \text{PreFilter}(\mathcal{T})$ ▷ Exclude irrelevant tokens, see Sec. 3.3
$\mathcal{G} \leftarrow \emptyset$
# Iterative Mining Loop
for i from 1 to N **do**
    # 1. Gradient-guided selection of promising candidates
    $\mathcal{T}_c \leftarrow \mathcal{T} \setminus (\mathcal{T}^* \cup \mathcal{G})$ ▷ Define current candidate set
    $\mathcal{B} \leftarrow \text{SelectCandidateBatch}(\mathcal{T}_c)$ ▷ See Sec. 3.1
    # 2. Robust verification of each candidate
    for each token $t \in \mathcal{B}$ **do**
        if VerifyIsGlitch(t) **then** ▷ See Sec. 3.2
            $\mathcal{G} \leftarrow \mathcal{G} \cup \{t\}$
        end if
        $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{t\}$ ▷ Mark token as processed
    end for
end for
return $\mathcal{G}$

exploration of the token space, which is particularly valuable for glitch token detection.

3 Methodology

Our goal is to develop a method that can automatically and efficiently discover glitch tokens in any given LLM. To this end, we propose **GlitchMiner**, a novel framework that formulates glitch token discovery as a behavior-driven optimization problem.

GlitchMiner systematically mines for these tokens through a powerful iterative loop, outlined in Algorithm 1. Each iteration first performs **Gradient-Guided Selection** to pinpoint tokens with the highest predicted output entropy. These candidates then immediately undergo **Robust Verification** to determine if they are genuine glitch tokens.

3.1 Gradient-Guided Candidate Selection

The core of GlitchMiner's discovery engine is the candidate selection process. To ensure the overall mining is efficient, this selection is not random; instead, it employs a local search strategy to intelligently find tokens that maximize model confusion, which we measure using output entropy.

Optimization Objective. To formally quantify a token's impact on model uncertainty, we calculate its output entropy within a repetition task. We use the following template for this optimization process:

User: Please repeat the string: "{token}"
Assistant: Sure, the string is: "{token}

Here, the first {token} represents the input token t being evaluated, and the second {token} is the next predicted token by LLM. To ensure that the input token appears directly as the model's next prediction, we prefill the assistant's response

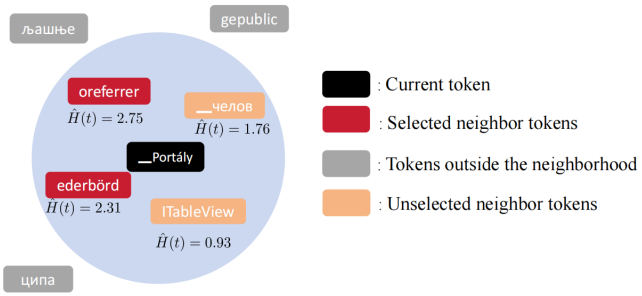

Figure 2: Visualization of GlitchMiner's local search process. The **pivot token** (black, t_c) serves as the reference point. Its **neighbor tokens** (orange and red) represent the $K = 4$ closest tokens in embedding space. Among these, the **candidate batch tokens** (red) are the top $B = 2$ tokens with the highest **approximate entropy values**, estimated via first-order Taylor approximation. Tokens outside the neighborhood (gray) are excluded to maintain approximation accuracy and computational efficiency.

with the phrase *Sure, the string is: "*. The entropy is computed based on the model's next-token distribution $p(v \mid \mathbf{h}(t))$, where $\mathbf{h}(t)$ denotes the prompt generated by inserting t into the repetition template h. The entropy is thus:

$$H(t) = -\sum_{v \in \mathcal{V}} p(v \mid \mathbf{h}(t)) \log p(v \mid \mathbf{h}(t))$$

Our goal is to find a batch of tokens $\mathcal{B}$ from the current set of available tokens $\mathcal{T}_c$ that maximizes the total entropy:

$$\mathcal{B} = \arg \max_{\mathcal{B} \subset \mathcal{T}_c, |\mathcal{B}| = B} \sum_{t \in \mathcal{B}} H(t)$$

where $\mathcal{T}_c = \mathcal{T} \setminus (\mathcal{T}^* \cup \mathcal{G})$, with $\mathcal{T}^*$ being the set of filtered (see Section 3.3) or previously verified non-glitch tokens, and $\mathcal{G}$ being the set of glitch tokens found.

Local Search Strategy. To efficiently solve this optimization problem, we introduce a **local search strategy** that addresses the limitations of global Taylor approximations. This process begins with an initial pivot token t_c and iteratively refines the search. In each step:

1. A local neighborhood $\mathcal{N}_K(t_c)$ is defined, consisting of the nearest neighbors K to the current pivot token t_c in the embedding space.
2. Within this neighborhood, the entropy of each candidate token $t \in \mathcal{N}_K(t_c)$ is estimated using a first-order Taylor approximation:
$$\hat{H}(t) \approx H(t_c) + \nabla_e H(t_c)^\top (e_t - e_{t_c})$$
where e_t and e_{t_c} are the respective embedding vectors. This approximation avoids the high cost of exact entropy calculations for all neighbors.
3. A batch $\mathcal{B}$ of B tokens with the highest approximated entropy $\hat{H}(t)$ is selected from the neighborhood. This batch is returned for verification.
4. To guide the next search step, the *actual* entropy H_t is computed for the tokens in $\mathcal{B}$, and the one with the highest actual entropy becomes the new pivot, t_c.

Figure 2 visualizes this process. This local search strategy significantly improves the accuracy of entropy estimation by focusing on tokens close to the pivot token. This entire process corresponds to the 'SelectCandidateBatch' function in Algorithm 1.

3.2 Robust Verification of Candidates

Once a batch of promising candidates is selected, each one undergoes a rigorous verification step ('VerifyIsGlitch' in our pipeline) to filter out false positives. A candidate is confirmed as a true glitch only if it fails the repetition task across a set of diverse prompt templates. To ensure this robustness, our verification process uses $m = 3$ templates for cross-validation: in addition to the template used for optimization (Sec. 3.1), we employ two others adapted from prior work (Land and Bartolo 2024a; Li et al. 2024). A token is formally classified as a glitch if and only if its failure rate (as defined in Equation 1) is 1. The additional prompt templates and analysis of false positives are provided in Appendix A.1 and Appendix A.2, respectively.

3.3 Practical Implementation: Search Space Pre-filtering

To maximize search efficiency, following (Land and Bartolo 2024b), we perform a one-time pre-filtering of the vocabulary ('PreFilter' in Algorithm 1). This step removes tokens that pose no real-world risk, allowing GlitchMiner's overall search to focus its resources. We filter three categories:

- **SPECIAL Tokens**: Predefined symbols like `[BOS]` or `</s>`. These are filtered out because they serve a reserved functional role for the model (e.g., indicating the start of a sequence) rather than representing user-generated text.
- **UNDECODEABLE Tokens**: Tokens that correspond to byte sequences that cannot be decoded into a valid string, often because they violate encoding standards like UTF-8. They are excluded as they do not map to any meaningful user-generated text.
- **UNREACHABLE Tokens**: Tokens that exist in the vocabulary but can never be generated by the model's tokenizer from any text input.

Further explanations of these token categories are provided in Appendix A.3.

4 Experiments

4.1 Experimental Setup

Evaluated LLMs. We used a diverse set of LLMs from five different model families to evaluate the performance of our glitch token detection approach. The selected models include Meta's Llama series (Touvron et al. 2023; AI 2024a), Alibaba's Qwen models (Yang et al. 2024; Alibaba 2024), Google's Gemma models (Team et al. 2024), Microsoft's Phi-3 models (Abdin et al. 2024), and Mistral models (Jiang et al. 2023; AI 2024b). The details are presented in Table 1.

Evaluation Metrics. We evaluate our glitch token detection method using the **Detected@N** metric, which counts the number of true glitch tokens identified within the top N predictions. For instance, Detected@1000 measures how many glitch tokens are found among the top 1000 candidates. This metric balances detection accuracy and query efficiency, reflecting a method's practical effectiveness under fixed query budgets. Comparing Detected@N values thus provides a direct measure of each method's ability to maximize glitch token discovery while minimizing computational resources, making it well-suited for real-world applications.

Baselines. We compare our proposed glitch token detection method with two state-of-the-art approaches: GlitchHunter (Li et al. 2024) and Magikarp (Land and Bartolo 2024a). These methods serve as the primary benchmarks for evaluating our approach.

Although GlitchProber (Zhang et al. 2024) is another relevant method, it follows a fundamentally different approach by pre-collecting a subset of glitch tokens to train a classifier, introducing a supervised learning component. In contrast, GlitchMiner, along with GlitchHunter and Magikarp, uses heuristic-based methods to detect glitch tokens without relying on labeled data or additional classifier training. This methodological difference makes a direct comparison less meaningful, so we focus our evaluation on methods that align more closely with our unsupervised approach.

Parameter Settings. In our implementation of GlitchMiner, we use K=32 and B=8 as the default parameters. These values were chosen based on empirical testing to balance computational efficiency and detection effectiveness. Specifically, K=32 defines the size of the local neighborhood considered in each iteration, while B=8 determines the batch size for entropy computation. These settings have shown to provide a good trade-off between exploration of the token space and exploitation of local information across various model architectures.

Initialization Strategy in Experiments. To ensure stable and consistent comparisons across runs, we initialize the search with the token exhibiting the smallest ℓ_2 norm in the embedding space, based on prior observations that such tokens often exhibit glitch-like behaviors. However, as shown in Figure 5, we found that GlitchMiner remains robust to different initialization choices, achieving similar performance even with random starting points.

4.2 Main Results

The performance of GlitchMiner against the baselines is detailed in Table 2. The results provide strong evidence for the effectiveness of our entropy-guided search, showing that GlitchMiner consistently discovers more glitch tokens than both GlitchHunter and Magikarp under fixed query budgets.

On average, GlitchMiner outperforms the strongest baseline, Magikarp, by 10.7% on the Detected@2000 metric. This performance advantage is not uniform but reveals an important trend: while Magikarp's simple norm-based heuristic can be effective for an initial, low-budget scan (e.g., on Llama-3.1-8B), GlitchMiner's more sophisticated local search strategy consistently proves more fruitful as the search budget expands. For instance, on Llama-2-7B-chat-hf, GlitchMiner finds nearly three times as many glitches as Magikarp (532 vs. 186), demonstrating its superior ability to uncover less obvious candidates that simple heuristics miss.

Model Family	Model Names
Llama Models	Llama-3.1-8B-Instruct , Llama-2-7B-chat-hf
Qwen Models	Qwen2.5-7B-Instruct , Qwen2-7B-Instruct
Gemma Models	Gemma-2-2b-it, Gemma-2-9b-it
Phi-3 Models	Phi-3-mini-128k-instruct, Phi-3.5-mini-instruct
Mistral Models	Mistral-7B-Instruct-v0.3, Mistral-Nemo-Instruct-2407

Table 1: Test LLMs used in the experiments.

Model	Metric	GlitchHunter	Magikarp	GlitchMiner (ours)
Llama-3.1-8B-Instruct	Detected@1000	25	**664**	568
	Detected@2000	56	935	**1164**
Llama-2-7B-chat-hf	Detected@1000	61	100	**319**
	Detected@2000	126	186	**532**
Qwen2.5-7B-Instruct	Detected@1000	75	1000	**1000**
	Detected@2000	180	**1893**	1839
Qwen2-7B-Instruct	Detected@1000	96	999	**1000**
	Detected@2000	191	1842	**1847**
Gemma-2-2b-it	Detected@1000	23	678	**744**
	Detected@2000	35	984	**1019**
Gemma-2-9b-it	Detected@1000	29	623	**775**
	Detected@2000	45	983	**1089**
Phi-3.5-mini-instruct	Detected@1000	20	393	**396**
	Detected@2000	44	496	**516**
Phi-3-mini-128k-instruct	Detected@1000	26	398	**404**
	Detected@2000	55	489	**517**
Mistral-7B-Instruct-v0.3	Detected@1000	6	110	**219**
	Detected@2000	19	130	**302**
Mistral-Nemo-Instruct-2407	Detected@1000	48	574	**695**
	Detected@2000	79	918	**976**
Average	Detected@1000	40.9	553.9	**612.0**
	Detected@2000	83.0	885.6	**980.1**

Table 2: Detected@1000 and Detected@2000 comparison of methods across different models.

These findings validate that framing glitch detection as a behavior-driven optimization problem is a more robust and generalizable strategy than relying on static, model-specific patterns.

4.3 Ablation Study

To evaluate the contributions of key components in GlitchMiner, we conducted ablation studies focusing on the local search strategy, neighborhood size K, batch size B, and initialization token.

Effect of Local Search. The local search strategy plays a crucial role in enhancing GlitchMiner's ability to detect glitch tokens by improving the precision of the Taylor approximation. Without local search, detection accuracy drops significantly (Figure 6), as global search lacks the necessary granularity to maintain precise approximations within the token space.

Effect of Neighborhood Size. We analyzed the impact of neighborhood size K on detection performance. As shown in Figure 3, increasing K generally leads to a decline in Detected@1000 values across models. This trend indicates that as K grows, the Taylor approximation becomes less effective, resulting in reduced prediction accuracy.

Effect of Batch Size. As shown in Figure 4, the performance of GlitchMiner remains relatively stable as batch size B increases. Notably, even with $B = 1$, GlitchMiner achieves effective detection results, indicating that it can make accurate predictions without relying on a large batch size.

Effect of Initialization Token. As shown in Figure 5, GlitchMiner's performance remains stable across different initialization tokens. The red dots represent the minimum ℓ_2 norm initialization, while the orange dots show three random trials. For most models, random initialization results are close to the minimum ℓ_2 norm, indicating that GlitchMiner achieves consistent detection accuracy regardless of the initialization approach.

4.4 Token Entropy Analysis

To further validate the effectiveness of our entropy-based approach in detecting glitch tokens, we conducted an entropy analysis comparing glitch tokens and normal tokens across different models. For each model, we computed the average entropy of glitch tokens (E_{Glitch}) and normal tokens (E_{Normal}).

Figure 7 presents the comparison of average entropy values

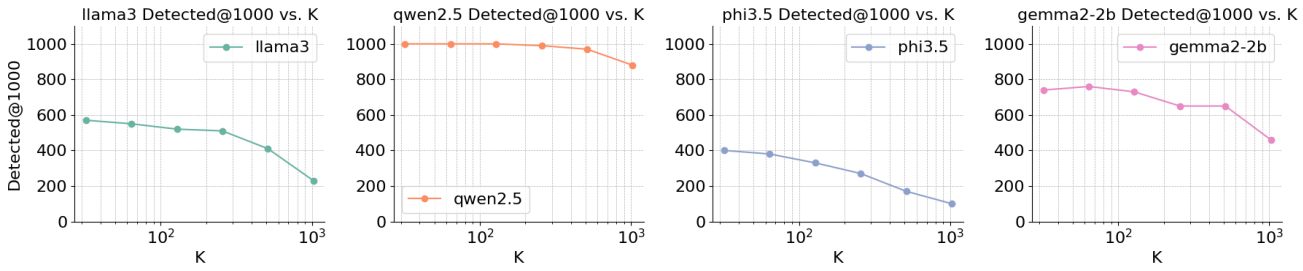

Figure 3: Impact of different Neighborhood Size K on GlitchMiner's performance

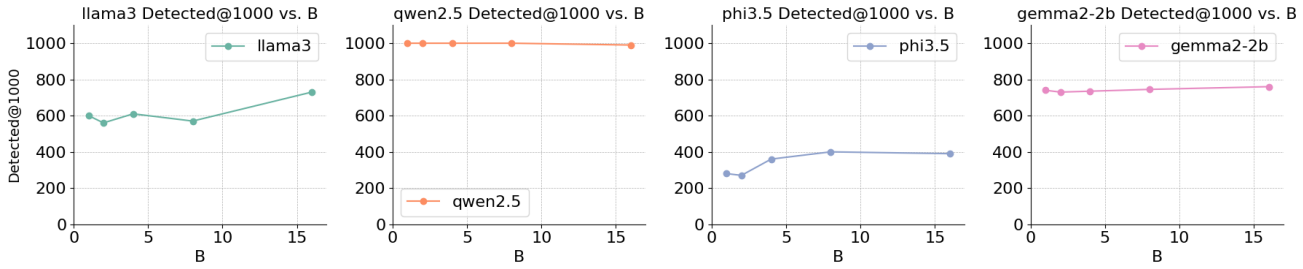

Figure 4: Impact of different Batch Size B on GlitchMiner's performance

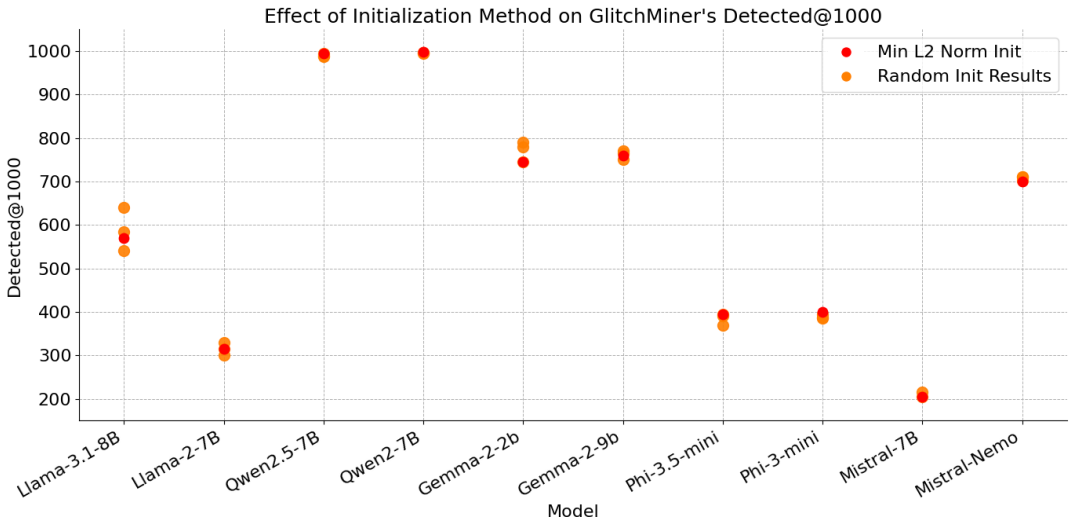

Figure 5: Effect of Initialization Method on GlitchMiner's Detected@1000 score.

between glitch tokens and normal tokens for each evaluated model. As shown in the figure, glitch tokens consistently exhibit significantly higher entropy than normal tokens across all models.

This pronounced difference in entropy values indicates that models are more uncertain when predicting glitch tokens compared to normal tokens. The higher entropy of glitch tokens validates our hypothesis that maximizing entropy effectively guides the search towards tokens that are challenging for the model to predict.

Moreover, the consistent pattern of higher entropy for glitch tokens across diverse model families—including Llama, Qwen, Gemma, Phi-3, and Mistral—demonstrates the generality and robustness of our entropy-based approach. This suggests that our method can be effectively applied to a wide range of LLMs with different architectures and tokenization strategies.

These findings validate the effectiveness of GlitchMiner's entropy-based optimization in efficiently detecting glitch tokens by focusing on areas of high prediction uncertainty within the model.

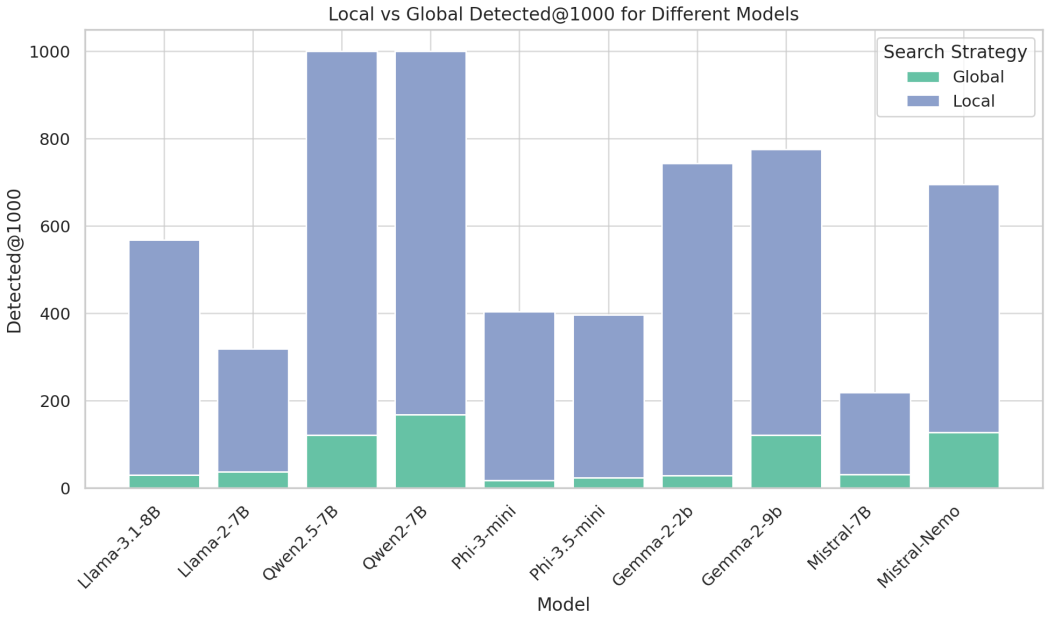

Figure 6: Comparison of GlitchMiner performance with and without local search strategy

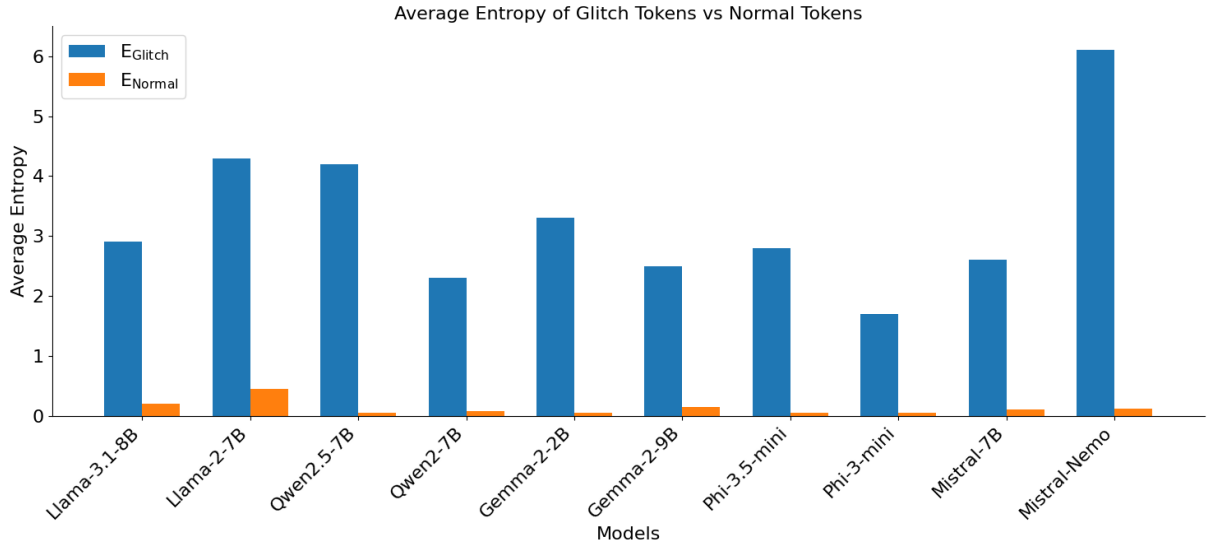

Figure 7: Average entropy comparison between glitch tokens and normal tokens across different models. Glitch tokens have higher entropy, indicating greater uncertainty in the model's predictions for these tokens.

5 Conclusion

In this paper, we introduced GlitchMiner, a novel framework that detects glitch tokens in LLM by reframing the problem as a behavior-driven optimization task. Departing from prior work that relies on static patterns, our approach identifies anomalous tokens by searching for inputs that maximize the model's predictive uncertainty, measured by output entropy. We operationalize this principle with an efficient gradient-guided local search strategy that can accurately pinpoint high-entropy candidates in the vast token space. Extensive experiments on 10 diverse LLMs demonstrated that GlitchMiner significantly outperforms state-of-the-art baselines in both detection accuracy and efficiency. Our work provides a more robust and generalizable tool for auditing and enhancing the reliability of LLMs, with future work pointing towards mitigation strategies and adaptation for black-box models.

Acknowledgments

We would like to express our sincere gratitude to Sander Land for his valuable insights and feedback on this work, par-

ticularly regarding the evaluation metrics. We wish to thank Guanlin Li for his valuable suggestions on our manuscript. This work was supported in part by the National Key R&D Program of China (2023YFB3107505), in part by Shaanxi Natural Science Funds for Distinguished Young Scholars(2023-JC-JQ-52), in part by the Natural Science Foundation of China (62302371), in part by the Postdoctoral Fellowship Program of CPSF (GZC20232035), and in part by the China Postdoctoral Science Foundation (2025M771552).

References

Abdin, M.; Jacobs, S. A.; Awan, A. A.; Aneja, J.; Awadallah, A.; Awadalla, H.; Bach, N.; Bahree, A.; Bakhtiari, A.; Behl, H.; et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

AI, M. 2024a. Introducing Llama 3.1: Our most capable models to date. https://ai.meta.com/blog/meta-llama-3-1/. Accessed: 2024-10-18.

AI, M. 2024b. Mistral Nemo: Advancing the Capabilities of Large Language Models. https://mistral.ai/news/mistral-nemo/. Accessed: 2024-10-18.

Alibaba. 2024. Qwen 2.5: Advancing AI for Everyone. https://qwen2.org/qwen2-5/. Accessed: 2024-10-18.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Ebrahimi, J.; Rao, A.; Lowd, D.; and Dou, D. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

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

Goel, A.; Gueta, A.; Gilon, O.; Liu, C.; Erell, S.; Nguyen, L. H.; Hao, X.; Jaber, B.; Reddy, S.; Kartha, R.; et al. 2023. Llms accelerate annotation for medical information extraction. In *Machine Learning for Health (ML4H)*, 82–100. PMLR.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Jiang, J.; Wang, F.; Shen, J.; Kim, S.; and Kim, S. 2024. A Survey on Large Language Models for Code Generation. *arXiv preprint arXiv:2406.00515*.

Jury, B.; Lorusso, A.; Leinonen, J.; Denny, P.; and Luxton-Reilly, A. 2024. Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian Computing Education Conference*, 77–86.

Land, S.; and Bartolo, M. 2024a. Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models. *arXiv preprint arXiv:2405.05417*.

Land, S.; and Bartolo, M. 2024b. Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models. *arXiv preprint arXiv:2405.05417*.

LessWrong Community. 2023. SolidGoldMagikarp (plus, prompt generation). https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/so. Accessed: 2023-09-25.

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. *Proceedings of the ACM on Software Engineering*, 1(FSE): 2075–2097.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Shen, X.; Chen, Z.; Backes, M.; Shen, Y.; and Zhang, Y. 2023. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *arXiv preprint arXiv:2308.03825*.

Shin, T.; Razeghi, Y.; Logan IV, R. L.; Wallace, E.; and Singh, S. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

Team, G.; Riviere, M.; Pathak, S.; Sessa, P. G.; Hardin, C.; Bhupatiraju, S.; Hussenot, L.; Mesnard, T.; Shahriari, B.; Ramé, A.; et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Traag, V. A.; Waltman, L.; and Van Eck, N. J. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1): 1–12.

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. *arXiv preprint arXiv:2403.18105*.

Wang, Y.; Ma, X.; and Chen, W. 2023. Augmenting black-box llms with medical textbooks for clinical question answering. *arXiv preprint arXiv:2309.02233*.

Wen, Y.; Jain, N.; Kirchenbauer, J.; Goldblum, M.; Geiping, J.; and Goldstein, T. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36.

Yang, A.; Yang, B.; Hui, B.; Zheng, B.; Yu, B.; Zhou, C.; Li, C.; Li, C.; Liu, D.; Huang, F.; et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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. *arXiv preprint arXiv:2408.04905*.

Zou, A.; Wang, Z.; Carlini, N.; Nasr, M.; Kolter, J. Z.; and Fredrikson, M. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.