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

Anonymous ACL submission

Abstract

Glitch tokens in Large Language Models (LLMs) are rare yet critical anomalies that can trigger unpredictable and erroneous model behaviors, undermining reliability and safety. Existing detection methods predominantly rely on predefined embedding or activation patterns, limiting their generalizability across diverse architectures and potentially missing novel glitch manifestations. We propose **GlitchMiner**, a gradient-based discrete optimization framework that identifies glitch tokens by maximizing prediction entropy to capture uncertainty, guided by a local search strategy for efficient token space exploration. Extensive evaluations on ten diverse LLM architectures demonstrate that GlitchMiner significantly outperforms state-of-the-art baselines in detection accuracy and efficiency. Our approach advances robust, architecture-agnostic glitch token detection, enhancing the security and trustworthiness of LLM-based applications in critical domains.

1 Introduction

Large Language Models (LLMs) have catalyzed breakthroughs across numerous domains, including code generation (Jiang et al., 2024; Chen et al., 2021; Nijkamp et al., 2022), healthcare (Goel et al., 2023; Wang et al., 2023), and education (Wang et al., 2024; Jury et al., 2024). Despite their success, a subtle yet critical vulnerability threatens their reliability: *glitch tokens*. Glitch tokens are anomalous token inputs that can induce erratic or nonsensical outputs, repetitive errors, or even bypass content filters, posing significant risks in sensitive applications (Geiping et al., 2024; LessWrong Community, 2023).

Prior work on glitch token detection (LessWrong Community, 2023; Li et al., 2024) often rely on heuristic analysis of token embeddings or clustering patterns. While effective in some settings, these pattern-based methods often struggle to generalize across the growing diversity of LLM architectures and may fail to detect newly emerging glitch tokens that deviate from known characteristics. This limitation calls for a more adaptive and behavior-driven detection methodology.

In this paper, we introduce **GlitchMiner**, a novel framework that directly leverages prediction behavior to identify glitch tokens through gradient-guided discrete optimization. Our key insight is to use entropy—a principled measure of prediction uncertainty—as the primary optimization objective. By maximizing entropy, GlitchMiner systematically uncovers tokens that destabilize model confidence without relying on fixed heuristics.

To address the challenge of discrete token search in a large vocabulary, we propose a local search strategy guided by first-order Taylor approximations of entropy changes in the embedding space. This approach improves optimization precision and reduces computational overhead, enabling efficient exploration.

Our contributions are summarized as follows:

- **Entropy-based detection paradigm**: Introducing entropy maximization as a robust and model-agnostic criterion for identifying glitch tokens.
- **Gradient-guided local search**: Combining gradient information with a localized search in embedding space to effectively and efficiently explore candidate tokens.
- **Cross-architecture adaptability**: Demonstrating consistent performance gains across ten LLMs from various families, including Llama, Qwen, Gemma, Phi, and Mistral models.

Evaluations on ten diverse LLM architectures demonstrate that GlitchMiner significantly outperforms state-of-the-art baselines in detecting glitch

tokens within fixed query budgets. Our approach enhances cross-architecture robustness and detection efficiency, advancing the security and reliability of LLM deployment in critical applications.

The remainder of this paper is organized as follows: Section 2 reviews related work on glitch token detection and gradient-based discrete optimization. Section 3 details the GlitchMiner methodology. Section 4 presents experimental evaluations and ablation studies. We conclude in Section 5 with discussions and future directions.

2 Background and Related Work

2.1 What is a Glitch Token?

Glitch tokens, a concept that gained attention following the discovery of the "SolidGoldMagikarp" phenomenon by Rumbelow et al. (LessWrong Community, 2023), are anomalous tokens in LLMs that can trigger unexpected and often erroneous behaviors when processed by the model. These tokens typically result from irregularities in the training process, such as underrepresentation in the training data or inconsistencies in tokenization (Land and Bartolo, 2024; Geiping et al., 2024). Glitch tokens have been found to exhibit several key characteristics:

- They often have abnormal embedding patterns, such as unusually small ℓ_2 norms or atypical positions in the embedding space (Land and Bartolo, 2024; LessWrong Community, 2023).
- When input to an LLM, they can cause the model to produce repetitive, nonsensical, or completely unrelated outputs (LessWrong Community, 2023; Li et al., 2024).
- They may lead to significant deviations in the model's internal activations compared to normal tokens (Zhang et al., 2024).
- Glitch tokens can sometimes bypass content filters or trigger unexpected model behaviors, potentially compromising the safety and reliability of LLMs (Geiping et al., 2024).

The study of glitch tokens is crucial for understanding and mitigating potential vulnerabilities in LLMs, as these tokens can potentially be exploited for adversarial attacks or lead to unintended model behaviors in critical applications (Geiping et al., 2024).

2.2 Glitch Token Detection

The detection of glitch tokens in LLMs has become an increasingly important area of research, given their potential to disrupt model performance and reliability. The fundamental approach to identifying these anomalous tokens often relies on repetition tasks, a method popularized by the SolidGoldMagikarp study (LessWrong Community, 2023). In these tasks, the model is prompted to repeat a given token, with the assumption that normal tokens should be easily reproducible, while glitch tokens often lead to failures in this simple task.

Building upon this basic principle, researchers have developed several sophisticated methods to detect glitch tokens more efficiently and accurately:

Magikarp (LessWrong Community, 2023) pioneered a fast and lightweight approach by analyzing token embeddings. It identifies potentially problematic tokens by examining characteristic patterns in the embedding space, such as unusual ℓ_2 norms. These candidate tokens are then verified using repetition tasks, allowing for quick detection through direct queries to the model.

GlitchHunter (Li et al., 2024) introduced a clustering-based method. This approach is founded on the observation that glitch tokens often cluster together in the embedding space. GlitchHunter constructs a Token Embedding Graph (TEG) to represent token relationships and applies the Leiden algorithm (Traag et al., 2019) to identify potential glitch clusters. These clusters are then refined through iterative hypothesis testing to improve detection accuracy.

GlitchProber (Zhang et al., 2024) takes a different approach by analyzing the internal activations within transformer layers, such as attention heads and hidden states. It reduces the dimensionality of these activations using PCA (Hotelling, 1933) and applies SVM classifiers (Cortes, 1995) to identify glitch tokens. Additionally, it integrates mitigation mechanisms by modifying neuron activations to minimize the impact of glitch tokens during inference.

While these methods have significantly advanced our ability to detect glitch tokens, they are inherently limited by their reliance on pattern-based heuristics. Magikarp and GlitchHunter focus on specific embedding patterns or clustering behaviors, while GlitchProber examines predefined patterns in activation spaces. This dependence on pattern-based approaches can constrain their adapt-

ability across diverse LLM architectures and potentially overlook novel forms of glitch tokens.

Our proposed method, GlitchMiner, aims to address these limitations by employing gradient-based discrete optimization to directly analyze an LLM's prediction behavior, potentially offering a more flexible and comprehensive approach to glitch token detection.

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 tokens to compute gradients and selects the token with the largest negative gradient to replace the current token, aiming to minimize 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 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 exploration of the token space, which is particularly valuable for glitch token detection.

3 Method

Algorithm 1 GlitchMiner Pipeline

Input: Token set $\mathcal{T}$, Iteration number I, Batch size B
Output: Glitch token set $\mathcal{G}$
Stage 1: Initialization
$\mathcal{T}^* \leftarrow \text{Filter}(\mathcal{T})$ # Filter out the unnecessary token set $\mathcal{T}^*$
$\mathcal{G} \leftarrow \emptyset$
# Stage 2: Mining
for $i \in I$ **do**
    Select a batch of tokens $\mathcal{B}$ from $\mathcal{T} \setminus (\mathcal{T}^* \cup \mathcal{G})$, where $|\mathcal{B}| = B$
    for each token $t \in \mathcal{B}$ **do**
        if Verify(t) **then**
            Add t to $\mathcal{G}$
        else
            Add t to $\mathcal{T}^*$
        end if
    end for
end for

GlitchMiner Pipeline. As illustrated in Algorithm 1, GlitchMiner consists of two main stages: initialization and mining.

During initialization, tokens that are unlikely to be glitch tokens are filtered out to reduce the search space and enhance computational efficiency. In the subsequent mining stage, batches of unverified tokens are iteratively selected for verification. Each token is evaluated: if identified as a glitch token, it is added to the discovered set; otherwise, it is excluded from further consideration. This iterative process continues until a predefined stopping criterion is met, such as reaching a target number of iterations or detecting a sufficient number of glitch tokens.

The key components of the GlitchMiner pipeline are detailed as follows:

1. **Token Filter**: Efficiently excluding tokens that do not require detection.

2. **Glitch Token Verification**: Procedures to confirm whether a token is a glitch token.

3. **Token Selection Strategy**: Approaches for selecting the most promising glitch token candidates at each iteration.

Through these components, GlitchMiner effectively implements the glitch token mining process, balancing thoroughness and efficiency in practice.

3.1 Token Filter

In the initialization stage of GlitchMiner, we first filter all tokens to remove those that do not need to be detected. Similar to (Land and Bartolo, 2024), we designed a token filter module that classifies tokens based on their characteristics and filters out those that are unnecessary for detection.

The core idea of the token filter module is to classify each token by decoding and then re-encoding it, ensuring it meets specific classification criteria. Specifically, we prepend a special prefix "«" to each token to maintain consistency during the encoding and decoding process. Then, we filter out tokens that fall into the following categories:

- **SPECIAL**: Special tokens, such as [BOS], </s>, etc., which are enclosed in brackets or angle brackets and contain alphabetic characters.
- **UNDECODEABLE**: Tokens that cannot be decoded, usually containing illegal characters.
- **UNREACHABLE**: Tokens that cannot be restored to their original token ID through the decoding and re-encoding process.

During the classification process, we first decode each token ID to obtain its corresponding string representation. If decoding fails, the token is classified as UNDECODEABLE. Next, we encode the decoded string and check if it can be restored to the original token ID. If it cannot, the token is classified as UNREACHABLE. If it meets the characteristics of a special token, it is classified as SPECIAL.

By filtering out these unnecessary tokens, we retain only the essential ones for subsequent mining, which improves the algorithm's efficiency and ensures the accuracy of the detection results.

3.2 Glitch Token Verification

Our glitch token verification process consists of two stages: optimization-driven detection and cross-verification.

Template for Optimization. For our primary detection and optimization process, we designed the following task template:

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

In this template, the blue text represents the prompt, the green text {token} is the input token being tested, and the red text {token} represents the model's predicted output. If the predicted token does not match the input token, it is initially classified as a potential glitch token.

This template is specifically designed to support our gradient-based optimization process. Its simplicity allows for clear gradient computations, while the prefilled assistant response ensures a uniform starting point for each token evaluation. Wrapping the token in "«" and "»" symbols minimizes interference from surrounding context. These features enable precise entropy calculations and gradient estimations, which are crucial for our entropy-guided search strategy described in the token selection section.

Cross-Verification. While our primary template is optimized for the search process, relying solely on one template may lead to false positives. To enhance the robustness of our detection, we implement a cross-verification step using two additional templates derived from GlitchHunter and Magikarp. Each potential glitch token identified by our primary template undergoes verification with these additional templates. A token is confirmed as a glitch token only if it fails the repetition task across all templates.

This two-stage approach combines the efficiency of our optimization-driven search with the reliability of multi-template verification, significantly reducing false positives while maintaining the effectiveness of our gradient-based detection method.

3.3 Token Selection

Motivation. Previous methods for glitch token detection often rely on manually observed embedding patterns. However, these approaches can be limited in their ability to adapt to diverse LLM architectures and may overlook novel forms of glitch tokens. To address these limitations, we propose an entropy-based approach. Entropy, as a measure of uncertainty in probability distributions, offers a model-agnostic way to identify tokens that cause unexpected behavior in LLMs. By focusing on tokens that maximize entropy, we can detect glitch tokens that deviate significantly from normal token behavior, regardless of their specific characteristics

or the underlying model architecture.

Entropy-Guided Exploration. We define the entropy $H(t)$ for a token t as:

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

where $\mathbf{h}(t)$ is the context embedding for token t, and $P(v \mid \mathbf{h}(t))$ is the model's predicted probability distribution over the vocabulary $\mathcal{V}$. By maximizing entropy, we aim to find tokens that cause high uncertainty in the model's predictions, potentially indicating glitch behavior.

Optimization Objective. Our goal is to find a batch of tokens $\mathcal{B}$ that maximizes the total entropy:

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

Here, $\mathcal{T}_c$ represents the current candidate set, defined as $\mathcal{T}_c = \mathcal{T} \setminus (\mathcal{T}^* \cup \mathcal{G})$, where:

- $\mathcal{T}$ is the initial set of all tokens in the model's vocabulary.
- $\mathcal{T}^*$ is the set of tokens that have been filtered out or verified as non-glitch tokens. This includes special tokens, undecoded tokens, and tokens that have been checked and confirmed as normal.
- $\mathcal{G}$ is the set of identified glitch tokens discovered so far in the process.

By excluding $\mathcal{T}^*$ and $\mathcal{G}$ from $\mathcal{T}$, we ensure that our search focuses only on the remaining unverified tokens, improving efficiency and preventing redundant checks.

Local Search Strategy. To address the limitations of global Taylor approximations and improve optimization efficiency, we propose a local search strategy. Starting from an initial token t_0, we compute its entropy $H(t_0)$ and gradient $\nabla_e H(t_0)$, then define a local neighborhood $\mathcal{N}_K(t_0)$ comprising the K nearest neighbors in the embedding space. Restricting the search to this neighborhood enhances approximation accuracy and enables focused exploration of promising glitch token candidates (see Figure 1).

For each candidate token $t \in \mathcal{N}_K(t_0)$, the entropy is estimated via first-order Taylor expansion:

$$\hat{H}(t) \approx H(t_0) + \nabla_e H(t_0)^\top (e_t - e_{t_0}),$$

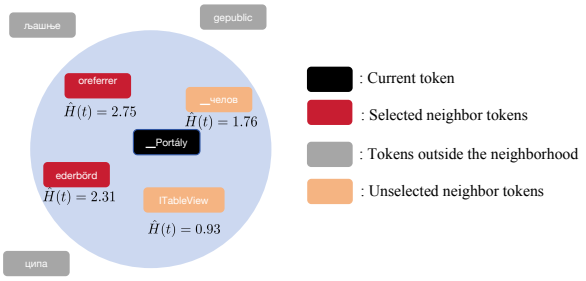

Figure 1: Visualization of GlitchMiner's local search process. The **current token** (black) serves as the reference point. Its **neighbor tokens** (orange and red) represent the $K = 4$ closest tokens in embedding space. Among these, the **candidate batch tokens** (red) are the top $B = 2$ tokens with 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.

where e_t, e_{t_0} are the embeddings of tokens t and t_0, respectively. This approximation allows efficient prediction of entropy changes without expensive exact evaluations.

From the neighborhood, a batch $\mathcal{B}$ of B tokens with the highest estimated entropy is selected. Actual entropy values are then computed for this batch, and the token with the maximum entropy becomes the new reference point for the next iteration. This iterative refinement ensures search progression toward tokens inducing higher model uncertainty.

By focusing on a localized region guided by entropy gradients, our approach mitigates the approximation errors inherent in global methods, balancing exploration and exploitation effectively. Consequently, it enables efficient and accurate identification of glitch tokens across diverse LLM architectures without relying on architecture-specific assumptions, enhancing robustness and generality.

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.

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.

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, 2024). 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

Table 2 displays the performance comparison of GlitchMiner with the state-of-the-art methods GlitchHunter and Magikarp across different LLM architectures using the Detected@N metric. The results highlight GlitchMiner's strong and versatile detection capabilities.

In the majority of cases, GlitchMiner achieved the highest Detected@2000 score, consistently outperforming both baselines in terms of identified glitch tokens within the top 2000 predictions. This is particularly evident in models such as Llama-2-7B-chat-hf, where GlitchMiner achieved a Detected@2000 of 532, surpassing Magikarp by a notable margin. For the Qwen and other models, GlitchMiner maintained robust performance, demonstrating adaptability across different LLM architectures.

These experimental findings underscore GlitchMiner's ability not only to accurately detect glitch tokens but also to generalize across a diverse set of model architectures. This adaptability and precision position GlitchMiner as a powerful tool for enhancing the robustness and security of LLMs.

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.

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	93.0	885.6	**980.1**

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

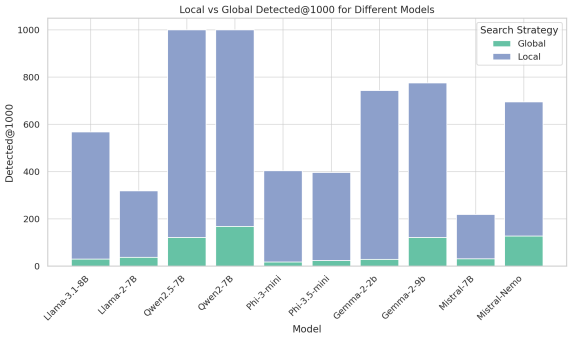

Figure 2: Comparison of GlitchMiner performance with and without local search strategy

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 2), 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 6 presents the comparison of average entropy values 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 fami-

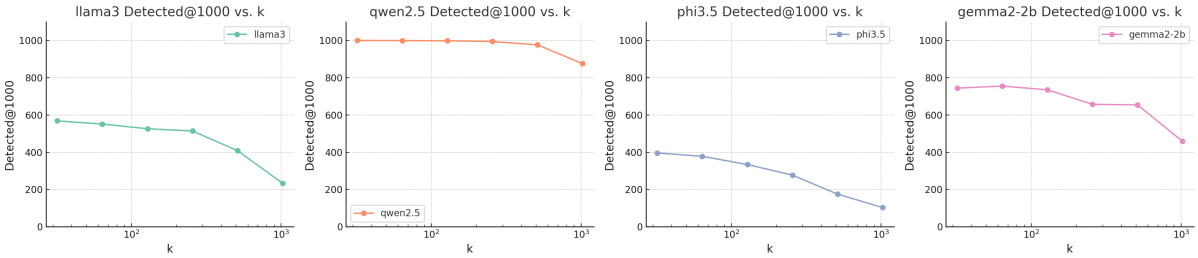

Figure 3: Impact of different Neighborhood Size K on GlitchMiner's performance

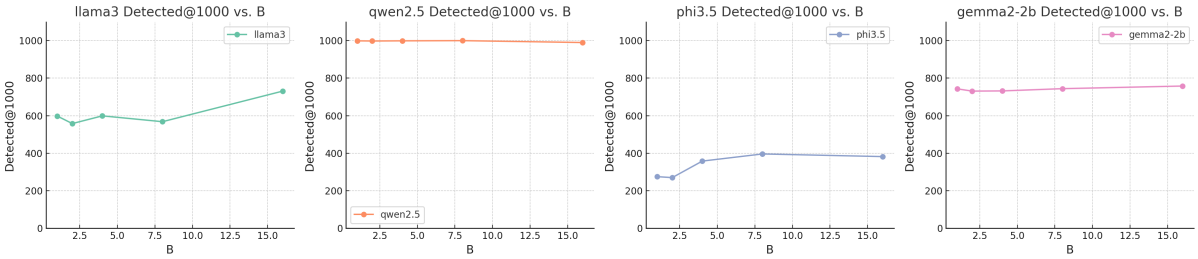

Figure 4: Impact of different Batch Size B on GlitchMiner's performance

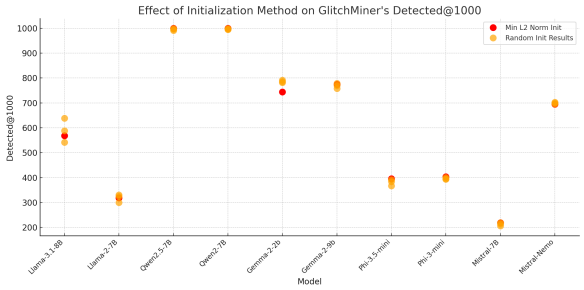

Figure 5: Effect of Initialization Method on GlitchMiner's Detected@1000 score.

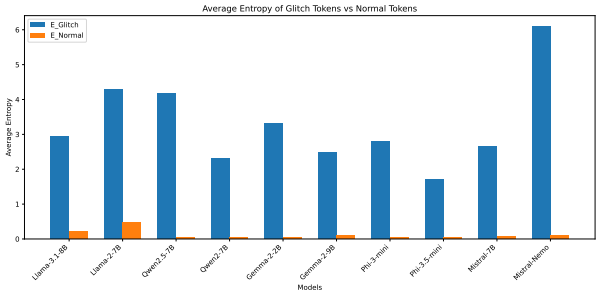

Figure 6: Average entropy comparison between glitch tokens and normal tokens across different models. Glitch tokens have higher entropy, indicating greater uncertainty in the model's predictions for these tokens.

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

5 Conclusion

In this paper, we introduced GlitchMiner, a novel framework for detecting glitch tokens in LLMs through gradient-based discrete optimization. Our method effectively combines entropy-based loss functions, which quantify the model's predictive uncertainty often associated with glitchy behavior, with a local search strategy that efficiently navigates the vast discrete token space. This synergy demonstrably improves both detection accuracy and computational efficiency. Experimental results across a diverse set of LLMs highlight GlitchMiner's robustness and versatility, consistently achieving superior glitch token detection rates under fixed query budgets compared to existing methods. Our findings confirm GlitchMiner's effectiveness in identifying glitch tokens across various architectures, underscoring its significant potential for enhancing the security and reliability of LLM-based applications, particularly in critical domains where such anomalies can pose substantial risks.

Limitations

GlitchMiner's core limitation is its reliance on gradient information, restricting its direct use to white-box models. Consequently, it is not directly applicable for end-users interacting with black-box APIs. However, this method is highly valuable for LLM developers and providers. With their intrinsic white-box access, they can effectively deploy GlitchMiner internally to proactively identify and understand glitch tokens, thereby enhancing model robustness and safety prior to public release. Future work might explore adapting its principles to gradient-scarce or black-box environments.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

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

Mistral AI. 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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Corinna Cortes. 1995. Support-vector networks. *Machine Learning*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. 2024. Coercing llms to do and reveal (almost) anything. *arXiv preprint arXiv:2402.14020*.

Akshay Goel, Almog Gueta, Omry Gilon, Chang Liu, Sofia Erell, Lan Huong Nguyen, Xiaohong Hao, Bolous Jaber, Shashir Reddy, Rupesh Kartha, et al. 2023. Llms accelerate annotation for medical information extraction. In *Machine Learning for Health (ML4H)*, pages 82–100. PMLR.

Harold Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian Computing Education Conference*, pages 77–86.

Sander Land and Max Bartolo. 2024. 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.

Yuxi Li, Yi Liu, Gelei Deng, Ying Zhang, Wenjia Song, Ling Shi, Kailong Wang, Yuekang Li, Yang Liu, and Haoyu Wang. 2024. Glitch tokens in large language models: categorization taxonomy and effective detection. *Proceedings of the ACM on Software Engineering*, 1(FSE):2075–2097.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *arXiv preprint arXiv:2308.03825*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12.

Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105*.

Yubo Wang, Xueguang Ma, and Wenhu Chen. 2023. Augmenting black-box llms with medical textbooks for clinical question answering. *arXiv preprint arXiv:2309.02233*.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Zhibo Zhang, Wuxia Bai, Yuxi Li, Mark Huasong Meng, Kailong Wang, Ling Shi, Li Li, Jun Wang, and Haoyu Wang. 2024. Glitchprober: Advancing effective detection and mitigation of glitch tokens in large language models. *arXiv preprint arXiv:2408.04905*.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.