Mitigating the Impact of Glitch Tokens via Targeted Retokenization

Muhammad Alif Al Hakim[1*], Dmitrii Kuzmin[1,2*], Carlos Javier Galán Zavala[3*], Tatsuya Hiraoka[1], Munachiso Samuel Nwadike[1], Ayana Niwa[1], Marco Cognetta[4], David Pohl [4], Kentaro Inui[1]

[1]MBZUAI, [2]Innopolis University, [3]Universidad de las Américas, [4]Institute of Science Tokyo

Correspondence: Muhammad.AlHakim@mbzuai.ac.ae

Abstract

This paper examines whether avoiding undertrained or "glitch" tokens and maximizing tokenization quality during inference can improve downstream task performance without retraining the model. We evaluate this question using a retokenization method that selects the highest-quality tokenization based on the trainedness of tokens, which is defined through the norm or entropy of token embeddings. We compare the performance difference between the tokenization that includes undertrained tokens and the one that avoids such tokens on multiple benchmarks. Our findings reveal a key nuance: undertrained tokens do not universally degrade performance; rather, degradation is conditioned on whether an undertrained token occupies semantically salient positions that are critical for solving the task. Consequently, we propose a targeted retokenization approach that preserves performance on ordinary inputs, while yielding consistent improvements when such tokens appear in high-importance contexts.

1 Introduction

LLM performance is fundamentally constrained by tokenization (Tamang and Bora, 2024; Seo et al., 2025). Because tokenizers are often trained separately from the model, a disconnect can emerge (Salesky et al., 2020; He et al., 2020; Hiraoka et al., 2020, 2021), creating undertrained or "glitch" tokens that are rarely seen during training (Cognetta et al., 2024a; Li et al., 2024). These tokens are known to degrade performance or cause undesirable behaviors during inference (Land and Bartolo, 2024).

In response to this issue, several methods have been developed to identify potential glitch tokens. Land and Bartolo (2024) introduce an indicator that analyzes the pretrained embedding space based on token proximity to known undertrained tokens to identify new undertrained tokens. As well, Wu et al. (2024) use gradient-based discrete optimization for LLM prediction analysis to identify and classify problematic glitch tokens.

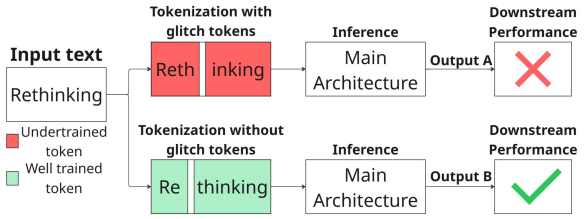


Figure 1: Overview of our experimental setup. The experiments were focused on changing performance by only modifying tokenization.

Building on this, an intuitive yet untested hypothesis emerges: **selecting tokenizations that avoid problematic tokens, as identified by token "trainedness", can improve downstream performance at inference time without tuning the model** (Figure 1). This paper empirically tests this hypothesis by employing non-canonical "retokenization." We also investigate under which conditions undertrained tokens undermine model performance. From our designed experiments, we found that undertrained tokens do not universally induce low-quality outputs; instead, performance collapse is triggered when undertrained tokens coincide with high-saliency information.

To mitigate this problem, we develop a retokenization method based on two existing trainedness metrics (Land and Bartolo, 2024; Wu et al., 2024). Our experimental results show that retokenization, which avoids undertrained tokens, improves performance when undertrained tokens appear in positions of high semantic salience and preserves performance when undertrained tokens occur in positions of low semantic salience or are absent from the input altogether.

Our targeted retokenization operates at inference time and requires no additional fine-tuning, making

* The first three authors contributed equally.

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 155–169
November 6-10, 2026 ©2026 Association for Computational Linguistics

it a lightweight and cost-effective way to handle realistic edge cases such as user typos or rare token combinations. As a result, it offers a viable approach to enhancing robustness in constrained settings where model retraining is impractical.

2 Retokenization Based on Trainedness

The purpose of retokenization is to avoid undertrained tokens. For example, given an input "Omnipotent Leader", which is tokenized into "Omni/**potent**/Lead/er" assuming the token "**potent**" is undertrained, our goal is to find an alternative tokenization candidate that is composed of well-trained tokens (Figure 2).

2.1 Trainedness-based Tokenizer

Let x be an input text and $\mathcal{Y}_V$ be a set of all possible tokenization candidates of x under the vocabulary V. For each tokenization candidate $y \in \mathcal{Y}_V$, let its constituent sequence of $|y|$ tokens be $y = (t_1, ..., t_{|y|})$. The goal of the trainedness-based tokenizer τ is to find $\hat{y}$ based on:

$$\hat{y} = \tau(x) = \arg\max_{y \in \mathcal{Y}_V} \phi(y), \quad (1)$$

where $\phi(y)$ is a metric to evaluate the trainedness of y. Intuitively, this method attempts to select the most well-trained tokenization candidate from all possible candidates.

The metric ϕ balances the trainedness score against a length penalty:

$$\phi(y) = \frac{\sum_{t \in y} f(t)}{|y|} \times \alpha - \mathcal{L} \times (|y| - 1), \quad (2)$$

where $\mathcal{L}$ is a scalar (e.g., 0.1) used to tune the penalty given to tokenizations with shorter tokens. The function $f(t)$ returns a trainedness score for the token t calculated by the following two options.

Norm-guided Metric ($\alpha = 1$): We use the L2 Norm of the token embedding as the trainedness score of the token $f(t)$, introduced as an indicator of undertrained tokens (Land and Bartolo, 2024).

Entropy-guided Metric ($\alpha = -1$): We use Token Entropy (Wu et al., 2024) as the trainedness score. This choice is motivated by the paper's finding of a substantial difference in entropy levels between undertrained and well-trained tokens.

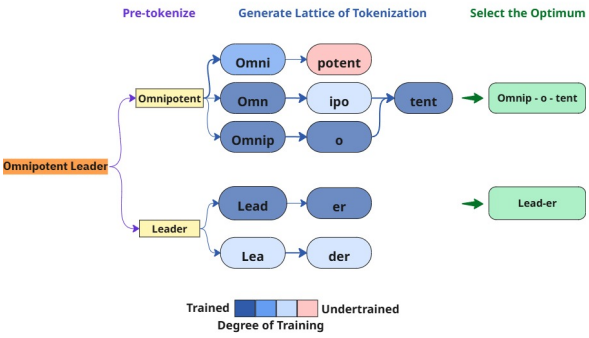


Figure 2: Conceptual overview of tokenization methodologies. Every pre-tokenized segment is re-evaluated to find a new tokenization that maximizes trainedness, using either norm- or entropy-guided metric.

2.2 Beam Search Strategy

Enumerating all possible tokenization candidates in Eq. (1) is impractical due to combinatorial explosion. Therefore, we instead adopt a beam search strategy that retains only the top K candidates (with $K = 5$), incrementally enumerating tokens from the beginning to the end while computing their trainedness scores.

2.3 Undertrained-aware Strategy

The trainedness-aware tokenizer explained above may change the canonical tokenization (i.e., the tokenization obtained by the original tokenizer) even if it does not include any undertrained tokens. In the case where we already know the corpus contains many apparently undertrained tokens, we can employ pre-filtering to select texts to be retokenized. Notably, if an input does not contain undertrained tokens, this method leaves its tokenization unchanged, even if alternatives with higher L2 norms or lower entropies exist.

3 Experiments on Datasets with Undertrained Tokens

3.1 Datasets

We first examine whether retokenization that avoids undertrained tokens improves performance when such tokens are present in the dataset. Because widely used, well-curated benchmarks rarely contain undertrained tokens, we construct two datasets that explicitly include them (Table 1). These settings are designed as stress tests, modeling high-stakes anomalies and possible scenarios, such as unusual or corrupted identifiers on social networking services, code switching, user typos, or input-level

Passkey Retrieval:
The secret password for the database is **krvldkf**. Extract the password and spell it by placing a hyphen between every letter. Output ONLY the hyphenated string. Do not provide any introductory text, labels, or explanations.
Modified GSM8K:
Rakou bought five cell phones for $150 each for a 3-month installment. A 2% interest will be charged for each unit. How much will **Rakou** pay each month for 3 months?

Table 1: Example of inputs with **undertrained token** in Passkey Retrieval and Modified GSM8K.

attacks by malicious users (Geh et al., 2025).

In both datasets, undertrained tokens appear in semantically crucial positions and must be correctly interpreted to solve the task. All experiments are conducted in a zero-shot setting with deterministic greedy decoding.

Passkey Retrieval: This task requires LLMs to spell out a "passkey" that is replaced with an undertrained token. Each prediction is evaluated using the Levenshtein distance between the generated output and the target token, normalized by the maximum character length of the two strings.

Modified GSM8K: We modify GSM8K (Cobbe et al., 2021) by replacing the nouns in each problem statement with an undertrained token. This substitution makes the undertrained tokens semantically relevant while leaving the underlying mathematical reasoning unchanged.

3.2 Tokenizers

We compare the downstream performance with two tokenization setups to validate the positive effect of retokenization.

Baseline Tokenizer: We use the original tokenizer bundled in the publicly available LLMs as a baseline tokenizer. The baseline may tokenize texts using undertrained tokens.

Trainedness-aware Tokenizer (Norm/Entropy): We used the Norm- and Entropy-guided approach (§2.1) as tokenizers. The undertrained-aware strategy (§2.3) was used only for experiments with undertrained tokens in the datasets (§3 and §4).

3.3 Models and Hardware

For our experiments, we selected four widely used models that were found to contain a significant number of undertrained tokens: Qwen2.5-7B-Instruct, Qwen2.5-32B-Instruct (Qwen et al., 2025), GLM-4.7-Flash (Team et al., 2025), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). All experiments were run on NVIDIA A40 x 2 GPUs.

Model	Dataset	Baseline	Norm	Entropy
Qwen2.5 7B (Instruct)	Passkey Re.	0.8447	**0.1639**	0.8030
	GSM8K	0.7901	**0.9019**	0.7901
Llama 3.1 8B (Instruct)	Passkey Re.	0.8995	**0.2644**	0.5298
	GSM8K	0.3468	**0.8274**	0.4084
Qwen2.5 32B (Instruct)	Passkey Re.	0.9241	**0.0647**	0.8400
	GSM8K	0.6494	**0.9331**	0.6859
GLM-4.7-Flash	Passkey Re.	0.9162	**0.0906**	0.7507
	GSM8K	0.8837	0.8730	**0.8905**

Table 2: Model Performance on datasets with high-salience undertrained tokens across tokenization schemes (Norm and Entropy-guided methods). For the passkey retrieval task, a smaller score means better performance.

3.4 Results

Table 2 presents performance differences across three tokenization methods. For the passkey retrieval task, the norm-based tokenizer substantially reduces error rates across all models compared to the baseline, while the entropy-based tokenizer yields smaller but positive performance gains. On the modified GSM8K, the norm-guided tokenizer notably improves most models, whereas entropy-based gains are marginal. Notably, GLM-4.7-Flash's high baseline suggests robust reasoning allows it to treat glitch tokens as abstract variables, bypassing the deficit.

Overall, these results indicate that retokenization aimed at avoiding undertrained tokens improves performance when such tokens appear in semantically salient positions, with the norm-guided method being particularly effective. We further discuss the empirical view of the difference in L2 Norm and entropy as our guiding metric in Appendix C.

4 Experiments on Datasets with Peripheral Undertrained Tokens

4.1 Contaminated Datasets

This section examines scenarios in which undertrained tokens appear in positions of low semantic salience. Specifically, we focus on cases where otherwise clean inputs are prefixed with redundant strings, such as those introduced by accidental copy-and-paste behavior in user queries.

Model	Dataset	Baseline	Norm	Entropy
Qwen2.5 7B (Instruct)	C-MMLU	0.6831	0.6824	**0.6834**
	C-WMT	38.2580	**38.3671**	38.3261
Llama 3.1 8B (Instruct)	C-MMLU	0.6142	**0.6345**	0.6341
	C-WMT	39.8344	**39.8551**	39.8152
Qwen2.5 32B (Instruct)	C-MMLU	**0.8024**	0.7952	0.8002
	C-WMT	40.2158	40.1933	**40.2288**
GLM-4.7-Flash	C-MMLU	**0.7317**	0.7115	0.7191
	C-WMT	40.4397	**40.6119**	40.5621

Table 3: Model Performance for evaluation with Peripherally Contaminated (represented as C-) Datasets.

To systematically simulate this setting, we construct contaminated variants of two widely used benchmark datasets by injecting undertrained tokens as prefixes to each input instance. These tokens serve as semantically irrelevant noise that should ideally be ignored during processing (Table 6). We consider two datasets corresponding to two representative NLP tasks:

Contaminated MMLU: We base this dataset on MMLU, a standard benchmark for multiple-choice question answering (Hendrycks et al., 2021). The contaminated version is obtained by prefixing each input with an undertrained token. We report accuracy as the evaluation metric.

Contaminated WMT: We similarly construct a contaminated version of WMT14 English–French (Bojar et al., 2014), which enables evaluation of generation tasks where tokenization effects are often more pronounced. We report SacreBLEU (Post, 2018) as the evaluation metric.

4.2 Results

Table 3 presents the experimental results on the peripherally contaminated datasets. Relative to their clean counterparts (Table 4 in §5), baseline performance is only slightly degraded by the presence of prefix tokens. The results also indicate that retokenization does not introduce adverse side effects in the case of larger models. In the case of smaller models, retokenization tends to alleviate performance degradation.

5 Experiments on Datasets without Undertrained Tokens

This section examines the effect of retokenization on datasets that do not contain undertrained tokens, utilizing the original versions of the MMLU and WMT benchmarks, which also serve as the base datasets for the contaminated setups described in Section 4. Table 4 presents the experimental results on these clean datasets. As shown in the table, the universal application of retokenization generally has a negative impact. The retokenization yields a positive result in only a single case, whereas the canonical tokenizer achieves the highest performance in all other cases.

Model	Dataset	Baseline	Norm	Entropy
Qwen2.5 7B (Instruct)	MMLU	**0.7027**	0.6782	0.6776
	WMT	**38.4789**	37.8030	37.2663
Llama 3.1 8B (Instruct)	MMLU	0.6172	**0.6419**	0.6008
	WMT	**39.8730**	37.3075	36.2037
Qwen2.5 32B (Instruct)	MMLU	**0.8027**	0.7934	0.8006
	WMT	**40.2306**	38.4055	39.1384
GLM-4.7-Flash	MMLU	**0.7336**	0.7055	0.7204
	WMT	**40.5478**	39.1722	38.6588

Table 4: Model Performance for the baseline evaluation on Clean Datasets (MMLU & WMT).

Model	Baseline	Norm	Entropy
Qwen2.5 7B (Instruct)	**677.3131**	650.5255	650.9193
Llama 3.1 8B (Instruct)	705.2956	**711.1760**	703.4217
Qwen2.5 32B (Instruct)	**692.3468**	679.3172	669.3751
GLM-4.7-Flash	**546.2234**	541.1923	532.7562

Table 5: Input throughput (tokens/s) comparison across tokenization methods (Canonical/Baseline, Undertrained-aware (Entropy), Undertrained-aware (Norm)) on Modified GSM8K dataset.

These results indicate that tokenization modification can degrade performance in certain cases. We attribute this behavior to the fact that, while the trainedness of individual constituent tokens may be moderately low, some multi-token combinations are themselves well trained. Retokenization can disrupt such well-trained combinations, which may in turn lead to performance degradation. However, these side effects can be avoided by a more targeted approach using the undertrained-aware strategy (Section 2.3), as this strategy leaves almost all tokenization in the baseline tokenizer unchanged.

6 Inference Latency

Because our method performs beam search over a tokenization lattice, we benchmark its computa-

tional overhead against the canonical tokenizer, as shown in Table 5. The generation speed remains unaffected, as retokenization only alters the input prior to forward pass. While input processing incurs a moderate overhead during lattice evaluation, the impact on total end-to-end latency is minimal. Furthermore, because of the undertrained-aware strategy (§2.3), this lattice-search overhead occurs only when undertrained tokens are present, preserving canonical inference latency for standard inputs.

7 Conclusion

In this study, we revisited the role of undertrained tokens in LLM performance and demonstrated that their impact is fundamentally conditioned on semantic context rather than being universally disruptive. By disentangling the effects of undertrained tokens across different levels of semantic salience, we clarified when retokenization can be beneficial.

When undertrained tokens appear in structurally important, high-salience positions, retokenization leads to consistent performance improvements. We showed that simple inference-time retokenization strategies, guided by the trainedness of token embeddings, can enhance reliability without requiring any model retraining.

When undertrained tokens occur in positions of low semantic salience, retokenization remains benign and does not introduce negative side effects. Although applying retokenization indiscriminately to clean inputs may degrade performance, this issue can be avoided by adopting the undertrained-aware strategy. Overall, our findings highlight that context-aware retokenization offers a practical and effective means of improving robustness to undertrained tokens, provided it is applied selectively to semantically critical contexts.

Limitations

We investigate the impact of “glitch tokens” on downstream performance, but we acknowledge several limitations and open questions for follow-up research.

Our study focuses strictly on inference-time interventions without parameter updates. While this highlights the potential of retokenization, it does not address the root cause of undertrained tokens during the pre-training phase.

The findings are based on Byte-Pair Encoding (BPE) tokenizers and models with fewer than 40 billion parameters; results may not generalize to other tokenization schemes or significantly larger models. Furthermore, our analysis is confined to English-language QA and machine translation tasks, making studying the impact of retokenization, especially on morphologically rich languages as important future work.

Additionally, our "High-Salience" datasets rely on synthetic injection (e.g., passkey retrieval) to guarantee sufficient interaction volume with rare tokens; further research is needed to quantify the natural occurrence rate of such high-stakes bottlenecks in organic user prompts.

Finally, while we utilized L2 norm and entropy as proxies for "trainedness," these are heuristic metrics; developing more rigorous, causally-derived indicators for token utility and quality remains an open challenge.

While our study was limited to four models and specific detection methods, our findings open several avenues for future research. Our findings suggest that the field needs to move beyond static, heuristic-based metrics for token quality and toward more dynamic, context-aware evaluations. Future work should investigate why different architectures exhibit unique token sensitivities and explore whether task-specific tokenization strategies could prove more effective than universal approaches.

Finally, our approach relies on specific metrics (L2 Norm, Token Entropy) as proxies for a token’s training level, based on existing work, and employs a re-tokenization through beam search strategy. Both of these represent potential avenues for future exploration with alternative methods.

Acknowledgement

The first three authors contributed equally to this work. Muhammad Alif Al Hakim designed and implemented the proposed retokenization method and strategy, developed the evaluation framework and its custom datasets, conducted the main experiments and performed analysis of its results, and drafted the manuscript. Carlos Javier Galan Zavala designed the retokenization method, assisted with experimental design, performed preliminary experiments and exploratory investigations, and drafted the manuscript. Dmitrii Kuzmin designed the retokenization method, performed preliminary experiments, and assisted with experimental design, retokenization method implementation, and manuscript revision. The remaining authors provided technical

advice across the paper's methodology and experiments, and assisted with manuscript revision. This work was conducted in part while the first three authors were participating in the MBZUAI Undergraduate Research Internship Program (UGRIP).

References

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ale s Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Marco Cognetta, Tatsuya Hiraoka, Rico Sennrich, Yuval Pinter, and Naoaki Okazaki. 2024a. An analysis of BPE vocabulary trimming in neural machine translation. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pages 48–50, Mexico City, Mexico. Association for Computational Linguistics.

Marco Cognetta, Vilém Zouhar, Sangwhan Moon, and Naoaki Okazaki. 2024b. Two counterexamples to tokenization and the noiseless channel. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16897–16906, Torino, Italia. ELRA and ICCL.

Marco Cognetta, Vilém Zouhar, and Naoaki Okazaki. 2024c. Distributional properties of subword regularization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10753–10763, Miami, Florida, USA. Association for Computational Linguistics.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.*, 20(3):273–297.

Renato Geh, Zilei Shao, and Guy Van Den Broeck. 2025. Adversarial tokenization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 20738–20765, Vienna, Austria. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Tatsuya Hiraoka. 2022. MaxMatch-dropout: Subword regularization for WordPiece. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4864–4872, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2020. Optimizing word segmentation for downstream task. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1341–1351.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2021. Joint optimization of tokenization and downstream model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 244–255.

Harold Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *Preprint*, arXiv:1804.10959.

Sander Land and Max Bartolo. 2024. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11631–11646, Miami, Florida, USA. Association for Computational Linguistics.

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. *Preprint*, arXiv:2404.09894.

Matt Post. 2018. A call for clarity in reporting bleu scores. *Preprint*, arXiv:1804.08771.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. Bpe-dropout: Simple and effective subword regularization. *Preprint*, arXiv:1910.13267.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li,

Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Elizabeth Salesky, Andrew Runge, Alex Coda, Jan Niehues, and Graham Neubig. 2020. Optimizing segmentation granularity for neural machine translation. *Machine Translation*, 34(1):41–59.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *Preprint*, arXiv:1508.07909.

Jean Seo, Jaeyoon Kim, SungJoo Byun, and Hyopil Shin. 2025. How does a language-specific tokenizer affect llms? *arXiv preprint arXiv:2502.12560*.

S Tamang and DJ Bora. 2024. Evaluating tokenizer performance of large language models across official indian languages. *arXiv e-prints*, pages arXiv–2411.

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1).

Zihui Wu, Haichang Gao, Ping Wang, Shudong Zhang, Zhaoxiang Liu, and Shiguo Lian. 2024. Glitchminer: Mining glitch tokens in large language models via gradient-based discrete optimization. *Preprint*, arXiv:2410.15052.

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. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 643–655. ACM.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

A Datasets

A.1 Clean Datasets

To assess the effectiveness of tokenization modification based on training degree, we chose two tasks: text classification and machine translation.

For the classification task, we selected MMLU (Hendrycks et al., 2021), a widely used multiple-choice question answering benchmark with 14,042 rows. MMLU was selected due to its suitability for assessing the difference in downstream performance resulting from tokenization changes on simple tasks.

For the machine translation task, we employed the French-to-English translation pair of WMT (Bojar et al., 2014), a well-established benchmark for this task with 3,003 rows. Tokenization's impact is significant for generation tasks, making it ideal for complex tasks requiring multiple tokens.

Both tasks are evaluated in a zero-shot setting. For MMLU, we report the percentage of correctly answered questions across all subjects. For WMT, we assess performance using the SacreBLEU metric (Post, 2018). All experiments are conducted with the Hugging Face and PyTorch libraries. In the MMLU experiments, we compute the likelihood of each option to determine the model's predicted

answer. In contrast, in WMT experiments, we use a greedy decoding strategy for translations. As the decoding strategy is deterministic, the experiments were run once. Figure 3 and 4 show the prompts that we used in evaluations for both tasks.

```
{question}
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}
Answer with the option letter.
```

Figure 3: Prompt used in MMLU evaluation.

```
Translate the following text from
{source_language} to {target_language}.
Provide only the translated text.


{source_language}: {source_text}
{target_language}:
```

Figure 4: Prompt used in WMT evaluation.

A.2 Contaminated Datasets

To more directly test our hypothesis, we created a synthetic, contaminated version of the clean datasets. This experimental setup is designed to create a clear scenario where glitch tokens are present but can be easily neglected without losing the meaning of a sentence.

We contaminated the dataset by prepending an undertrained token to the prompt as a semantically irrelevant prefix, ensuring it acted as noise rather than instruction. This contamination was tailored to each model, as the set of undertrained tokens differs for each model. Specifically, these undertrained tokens were filtered to include only Latin-derived tokens of at least four characters in length.

Table 6 illustrates the structural format of the contaminated datasets.

A.3 Forcing Undertrained Tokens Into Saliency

Since LLMs are robust to noise, this approach was designed so that the glitch tokens could not be treated as noise without losing the meaning of the sentence. This new approach brings us another problem. If we replace important fragments in each sentence of the aforementioned datasets with glitch tokens, the sentence itself might lose meaning, regardless of the effect this glitch token has

Dataset	Example
MMLU	ForCanBeConverted {question} ...
WMT	ForCanBeConverted Translate the following text...

Table 6: Structural layout of the contaminated MMLU and WMT datasets. In this case, "ForCanBeConverted" is the undertrained token prepended to the instance.

```
PROMPT_TEMPLATES = {
    "1": "The secret password for the database is 'TOKEN'.
          Extract the password and spell it by placing a
          hyphen between every letter.\nOutput ONLY the
          hyphenated string. Do not provide any introductory
          text, labels, or explanations.",
    "2": "I was walking down the street and saw a sign that
          said 'TOKEN' in big neon letters.\nExtract the word
           written on the sign and spell the word with a
          hyphen between every letter. Constraint: Output
          ONLY the final hyphenated string. No other words.",
    "3": "Extract the value of 'id' from this object: {{'id':
           'TOKEN', 'status': 'active'}}. Extract the value
          of 'id' and spell it by placing a hyphen between
          every letter.\nOutput ONLY the hyphenated string.
          Do not provide any introductory text, labels, or
          explanations.",
    "4": "Please listen carefully. The weather is nice. The
          sky is blue. The target code is 'TOKEN'. The grass
          is green. Extract the target code and spell it by
          placing a hyphen between every letter.\nOutput ONLY
           the hyphenated string. Do not provide any
          introductory text, labels, or explanations.",
    "5": "The system generated a unique identifier: 'TOKEN'.
           Extract the identifier code and spell it by
          placing a hyphen between every letter.\nOutput ONLY
           the hyphenated string. Do not provide any
          introductory text, labels, or explanations."
}
```

Figure 5: Prompt used for contextually salient undertrained tokens.

on the model, thereby skewing the benchmark results. In order to avoid this problem, we created two solutions.

First, we based our approach on the prompt engineering used by (Land and Bartolo, 2024) to create a prompt meant to force the undertrained tokens into a contextually relevant sentence position. This is done by phrasing the token as a "passkey" and requesting the model to retrieve and spell it out. There are five types of prompts that we used in this experiment, which can be seen in Figure 5.

Where the undertrained token takes the place of the password. The metric on this evaluation (error ratio) is the levenshtein distance between the undertrained token and the model's output (minus the hyphens) divided by the length of the longest string between the two. Therefore, the best performance is 0, and the worst performance is 1.

The second approach was using "GSM8K", a dataset containing math reasoning problems. This dataset was used because nouns could be safely replaced without altering the result of the math itself. To identify words suitable for replacement with undertrained tokens, we employ Gemini 2.5 Flash Lite. The prompt used for this process is provided in Figure 6.

```
You are an expert Computational Linguist and Data
      Augmentation Specialist. Your task is to
      analyze math word problems from the GSM8K
      dataset and identify a single "Safe Target
      Entity" that can be replaced with a nonsense
      word without breaking the mathematical logic or
       semantic structure of the problem.

### GOAL
Identify one specific noun or proper noun in the
      text that acts as a "Surface Entity" (a person,
       object, or place) and is NOT a mathematical
      variable, unit of measurement, or logical
      operator.

### SELECTION
1. **Proper Names**: Names of people or specific
      places (e.g., "Natalia", "James", "Seattle").
      These are always safe to replace.
2. **Concrete Physical Objects**: Items being
      counted or manipulated (e.g., "apples", "clips
      ", "toys", "cards").
3. **Generic Roles**: People described by role (e.g.,
       "friends", "students", "customers").

### STRICT EXCLUSION LIST (Do NOT Select)
- **Math Terms:** total, sum, difference, product,
      quotient, remainder, average, mean, median,
      ratio.
- **Value Terms:** cost, price, amount, value, rate,
       speed, capacity, weight, height, distance.
- **Units of Measurement:** hours, minutes, seconds,
       days, weeks, months, years, dollars, cents,
      degrees, miles, km, kg, lbs.
- **Logical Connectors:** part, piece, half, third,
      quarter, times.

### OUTPUT FORMAT
You must return a valid JSON object with the
      following fields:
- "target_word": The exact substring from the
      original text to be replaced. If multiple valid
       options exist, pick the safest one (highest
      priority).
- "is_safe": Boolean (true/false).
- "reason": A brief explanation of why this word is
      safe or why no safe word was found.

If NO safe word exists (e.g., the problem is purely
      abstract like "What is 5 + 5?"), return "
      target_word": null.
```

Figure 6: Prompt used for detecting words that can safely be changed with an undertrained token.

B Analysis on Peripherally Contaminated Experiments

To further investigate the effect of undertrained tokens on low-saliency positions, we analyzed the impact of tokenization input changes resulting from retokenization approaches. As shown in Figures 7 and 8, the majority of questions in the contaminated MMLU set exhibited no change in score, indicating that the model is robust to undertrained tokens in these positions.

In the contaminated WMT experiments, Figures 9 and 10 reveal that the generative task is more sensitive; instances with no change in SacreBLEU score occur less frequently, although they still constitute the majority of cases. Furthermore, in certain instances, retokenization can degrade generation quality more severely than the presence of an undertrained token in a low-saliency position.

Additionally, we analyzed the impact of these low-saliency undertrained tokens on score variations using the canonical tokenizer. This comparison isolates the direct effect of the undertrained tokens themselves. As illustrated in Figure 11, the SacreBLEU scores remain unchanged in the majority of cases after exclusively introducing low-saliency undertrained tokens. Across all conducted analyses, larger models consistently demonstrate greater robustness, as evidenced by a higher proportion of "no change" outcomes and a minimal occurrence of degraded performance. These findings indicate that the presence of an undertrained token in low-saliency positions does not necessarily result in lower-quality outputs.

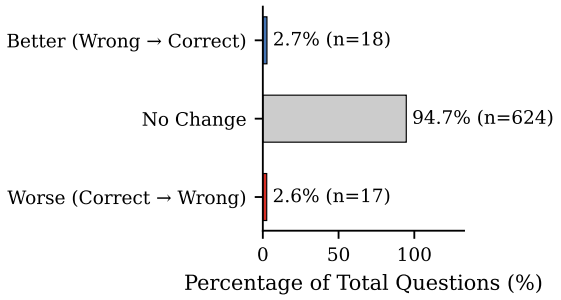
Model	Metric	Spearman (r_s)
Llama 3.1 8B	L2 Norm	-0.521
	Negative Entropy	0.298
Qwen 2.5 7B	L2 Norm	-0.264
	Negative Entropy	0.146
Qwen 2.5 32B	L2 Norm	-0.191
	Negative Entropy	-0.095
GLM-4.7 Flash	L2 Norm	-0.243
	Negative Entropy	0.115

Table 7: Spearman correlation (r_s) between model trainedness metrics and performance.

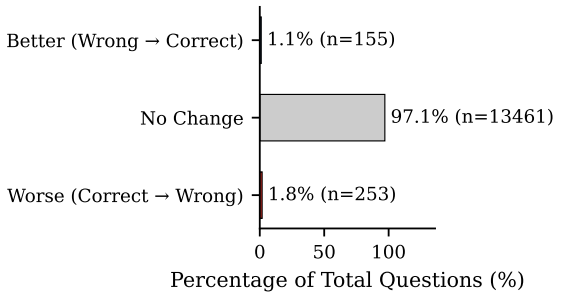
C Trainedness Metric Correlation

To analyze our trainedness heuristic indicators, we assessed the correlation between a token's heuristic score and its downstream performance. Specifically, we calculated the average error rate for each undertrained token in the Passkey Retrieval task using the canonical tokenizer. We then employed Spearman's rank correlation to analyze the relationship between each token's heuristic score and its corresponding average error rate.

Table 7 presents the Spearman rank correlation analysis across the evaluated models. As shown,

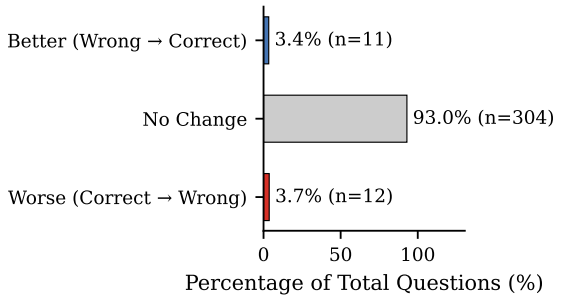


(a) Qwen2.5-7B-Instruct

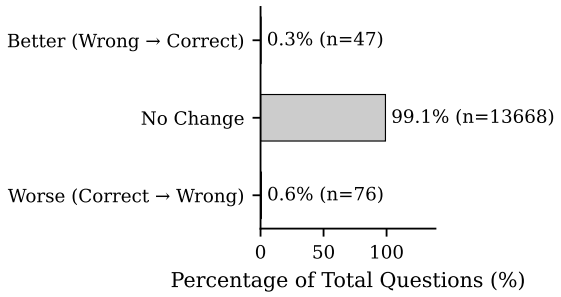


(b) Qwen2.5-32B-Instruct

Figure 7: Distribution of score changes on the contaminated MMLU set following **norm**-guided retokenization for (a) Qwen2.5-7B-Instruct and (b) Qwen2.5-32B-Instruct. The categories represent changes in answer accuracy: "Better", "No Change", and "Worse".



(a) Qwen2.5-7B-Instruct



(b) Qwen2.5-32B-Instruct

Figure 8: Distribution of score changes on the contaminated MMLU set following **entropy**-guided retokenization for (a) Qwen2.5-7B-Instruct and (b) Qwen2.5-32B-Instruct. The categories represent changes in answer accuracy: "Better", "No Change", and "Worse".

the L2 Norm consistently exhibits a negative correlation with the baseline error rate across all architectures. This inverse relationship indicates that a reduction in a token's embedding norm corresponds to an increased likelihood of task failure. Conversely, token entropy displays an inconsistent correlation across models, yielding a positive correlation in several instances. This suggests that tokens flagged as highly trained by the entropy metric may correspond to higher downstream error rates, thereby explaining why the entropy-guided tokenizer failed to match the performance gains achieved by the norm-guided tokenizer.

D Related Work

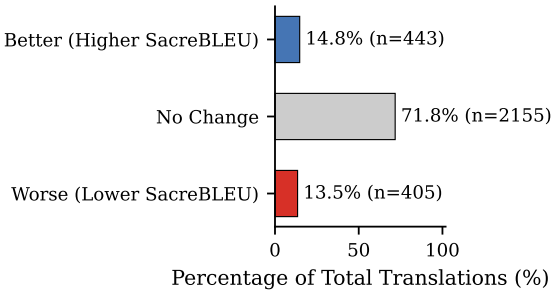
LLMs primarily rely on subword tokenization algorithms to manage vocabulary size and handle rare words. The two most prominent approaches are Byte Pair Encoding (BPE), a deterministic method that iteratively merges the most frequent character pairs (Sennrich et al., 2016), and the Unigram model, a probabilistic approach that samples from multiple potential subword segmentations (Kudo, 2018). A critical, shared characteristic of these methods is that the tokenizer's training is separate from the main language model's training. This disconnect is a primary source of "glitch" or "undertrained" tokens (Li et al., 2024): the vocabulary may contain valid tokens, but the model fails to develop a robust representation for some reasons, such as infrequent appearance in the training data.

Since such tokens cause unexpected behavior of the pretrained LM, they may damage downstream performance. Several methods have been developed to detect undertrained or "glitch" tokens, significantly advancing our understanding of tokenization failures. A prominent line of work operates on the observation that such tokens cluster together within a model's embedding space. For instance, "Fishing for Magikarp" first identifies a seed set of undertrained tokens and then measures their proximity to other vocabulary items using cosine similarity or L2 distance to determine their training level (Land and Bartolo, 2024). Similarly, "Glitch Hunter" constructs a graph of the embedding space and applies the Leiden clustering algorithm (Traag et al., 2019) to isolate entire "glitch clusters" (Li et al., 2024).

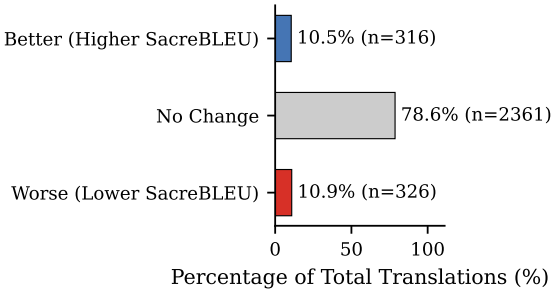
Moving beyond the embedding space, other methods analyze the model's internal dynamics. "Glitch Prober" analyzes the activation states of a Transformer's attention heads and hidden layers, using PCA (Hotelling, 1933) for dimensionality reduction and an SVM (Cortes and Vapnik, 1995) classifier to identify undertrained tokens (Zhang et al., 2024). While effective, the aforementioned approaches rely on pattern-based heuristics, which may not generalize properly across different model architectures. To address this limitation, "Glitch Miner" proposes a more direct, model-agnostic approach by using gradient-based discrete optimization to analyze the LLM's final prediction behavior (Wu et al., 2024).

Another line of research demonstrates that introducing stochasticity into the tokenization process can serve as a powerful form of regularization, improving model robustness and performance. This approach, often termed "subword regularization," treats tokenization not as a fixed, deterministic step but as a source of data augmentation (Kudo, 2018). For example, "BPE-Dropout" improves downstream performance by randomly dropping BPE merge operations during training (Provilkov et al., 2020). Similarly, "MaxMatch Dropout" achieves gains in machine translation and text classification by randomly disabling states in the WordPiece vocabulary trie (Hiraoka, 2022). Other work has explored the benefits of uniform sampling from a tokenization lattice, again showing improved translation quality (Cognetta et al., 2024c). Collectively, these studies demonstrate that significant performance gains can be achieved by modifying only the input tokenization module, making the model more resilient to segmentation variations.

A significant limitation of the previously discussed approaches is their exclusive focus on extrinsic metrics, such as downstream benchmark performance. The intrinsic quality of the tokenizer itself is rarely the subject of evaluation in the literature. One notable exception is Rényi efficiency, an information-theoretic metric that measures how effectively a tokenizer compresses text, which has been shown to correlate with task performance (Zouhar et al., 2023). Specifically, it quantifies the average number of bits required to represent each character in a text after tokenization. A lower value signifies greater compression, suggesting the tokenizer is using its vocabulary more efficiently. Despite its utility, Rényi efficiency is a flawed metric, as arbitrarily optimizing for better compression can paradoxically lead to a decrease in extrinsic performance (Cognetta et al., 2024b). This highlights the need for more robust intrinsic evaluation methods.



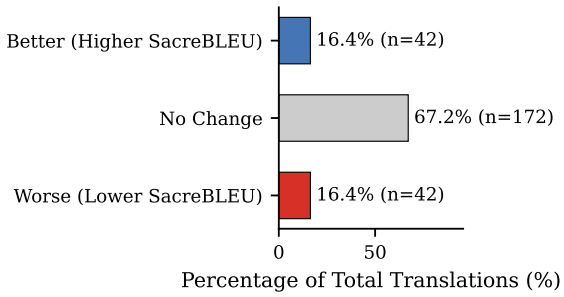
(a) Qwen2.5-7B-Instruct



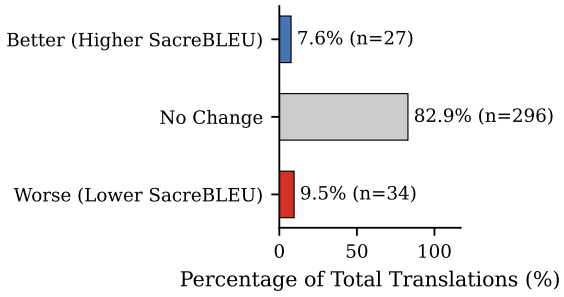
(b) Qwen2.5-32B-Instruct

Figure 9: Distribution of score changes on the contaminated WMT set following **norm**-guided retokenization for (a) Qwen2.5-7B-Instruct and (b) Qwen2.5-32B-Instruct. The categories represent changes in answer quality: "Better", "No Change", and "Worse"

Prior work has established methods for detecting undertrained tokens and has shown that modifying the tokenization process itself can regularize models. However, the evaluation of tokenizer quality remains heavily reliant on expensive, extrinsic benchmarks. Despite these advancements in detection, the intuitive hypothesis that simply avoiding glitch tokens at inference time improves downstream performance remains largely unverified. This paper addresses this critical gap by conducting a systematic empirical evaluation of the glitch token avoidance hypothesis across a diverse range of tasks and datasets.

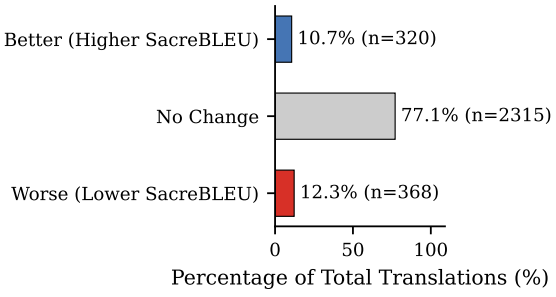


(a) Qwen2.5-7B-Instruct

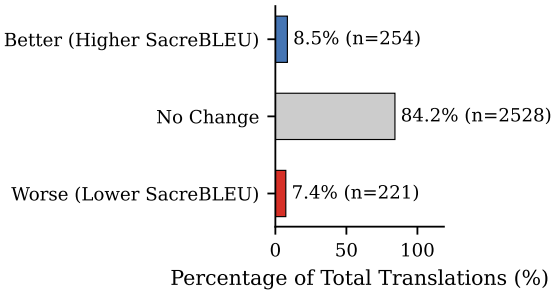


(b) Qwen2.5-32B-Instruct

Figure 10: Distribution of score changes on the contaminated WMT set following **entropy**-guided retokenization for (a) Qwen2.5-7B-Instruct and (b) Qwen2.5-32B-Instruct. The categories represent changes in answer quality: "Better", "No Change", and "Worse"



(a) Qwen2.5-7B-Instruct



(b) Qwen2.5-32B-Instruct

Figure 11: Distribution of translation quality changes (SacreBLEU) after introducing a low-saliency undertrained token to each instance in the clean WMT dataset for (a) Qwen2.5-7B-Instruct and (b) Qwen2.5-32B-Instruct. Performance changes are categorized as "Better" (increased score), "No Change" (identical score), and "Worse" (decreased score).

E Additional Findings

E.1 Performance Impact Across Diverse Domains

Figure 12 illustrates the performance variations when applying a trainedness-based tokenizer (§2.1) to the clean MMLU dataset using the Qwen2.5-7B-Instruct model. Although the effects vary across different subjects and categories, the majority of subjects experience a degradation in performance. Furthermore, there is no discernible pattern indicating which specific subject domains are systematically degraded or benefited by this retokenization.

E.2 GSM8K Generation Examples

Table 8 presents an example of responses from each evaluated model on the modified GSM8K experiment. GLM-4.7-Flash demonstrates robust performance through its ability to utilize abstract variables as substitutes for the glitch or undertrained tokens. In contrast, alternative models either truncate their outputs prematurely, immediately before generating the glitch token, or lose track of the mathematical objective during the reasoning process.

E.3 Tokenization Examples

Here are several examples of tokenizations from our experiments with the WMT dataset. Each entry shows the tokenization applied, the corresponding output or translation, and the resulting SacreBLEU score.

E.3.1 Example 1

In this example, the first entry presents the tokenization produced by the baseline tokenizer on the clean WMT dataset, while the second entry displays the tokenization from the baseline tokenizer on the contaminated WMT dataset. This scenario demonstrates that adding an undertrained token does not affect the output.

Tokenization: [’ĠThe’, ’Ġusually’, ’Ġdull’, ’Ġarena’, ’Ġof’, ’Ġhighway’, ’Ġplanning’, ’Ġhas’, ’Ġsuddenly’, ’Ġspawned’, ’Ġintense’, ’Ġdebate’, ’Ġand’, ’Ġcolorful’, ’Ġalliances’]

Translation: L’arène habituellement terne de la planification des autoroutes a soudainement donné lieu à un débat intense et à des alliances colorées

SacreBLEU: 15.11823598113958

Tokenization: [’Ġen’, ’CppMethodPointer’, ’Ġ–’, ’ĠThe’, ’Ġusually’, ’Ġdull’, ’Ġarena’, ’Ġof’, ’Ġhighway’, ’Ġplanning’, ’Ġhas’, ’Ġsuddenly’, ’Ġspawned’, ’Ġintense’, ’Ġdebate’, ’Ġand’, ’Ġcolorful’, ’Ġalliances’]

Translation: L’arène habituellement terne de la planification des autoroutes a soudainement donné lieu à un débat intense et à des alliances colorées

SacreBLEU: 15.11823598113958

E.3.2 Example 2

In this example, both entries present tokenizations from the contaminated WMT dataset. The first entry uses the baseline tokenizer, while the second entry uses an undertrained-aware tokenizer. This scenario provides examples that avoiding an undertrained token does not alter the output.

Tokenization: [’Ġin’, ’artisanlib’, ’Ġ–’, ’ĠThousands’, ’Ġof’, ’Ġmotorists’, ’Ġhave’, ’Ġalready’, ’Ġtaken’, ’Ġthe’, ’Ġblack’, ’Ġboxes’, ’,’, ’Ġsome’, ’Ġof’, ’Ġwhich’, ’Ġhave’, ’ĠGPS’, ’Ġmonitoring’, ’,’, ’Ġfor’, ’Ġa’, ’Ġtest’, ’Ġdrive’]

Translation: Des milliers de conducteurs ont déjà emprunté les boîtes noires, certaines desquelles sont équipées de suivi GPS, pour une démonstration.

SacreBLEU: 18.465221092138684

Tokenization: [’Ġi’, ’na’, ’rt’, ’i’, ’san’, ’lib’, ’Ġ–’, ’ĠThousands’, ’Ġof’, ’Ġmotorists’, ’Ġhave’, ’Ġalready’, ’Ġtaken’, ’Ġthe’, ’Ġblack’, ’Ġboxes’, ’,’, ’Ġsome’, ’Ġof’, ’Ġwhich’, ’Ġhave’, ’ĠGPS’, ’Ġmonitoring’, ’,’, ’Ġfor’, ’Ġa’, ’Ġtest’, ’Ġdrive’]

Translation: Des milliers de conducteurs ont déjà emporté les boîtes noires, certaines desquelles sont équipées de suivi GPS, pour une démonstration.

SacreBLEU: 18.465221092138684

E.3.3 Example 3

Similar to the first example (§E.3.1), but in this example, adding an undertrained token results in a higher score.

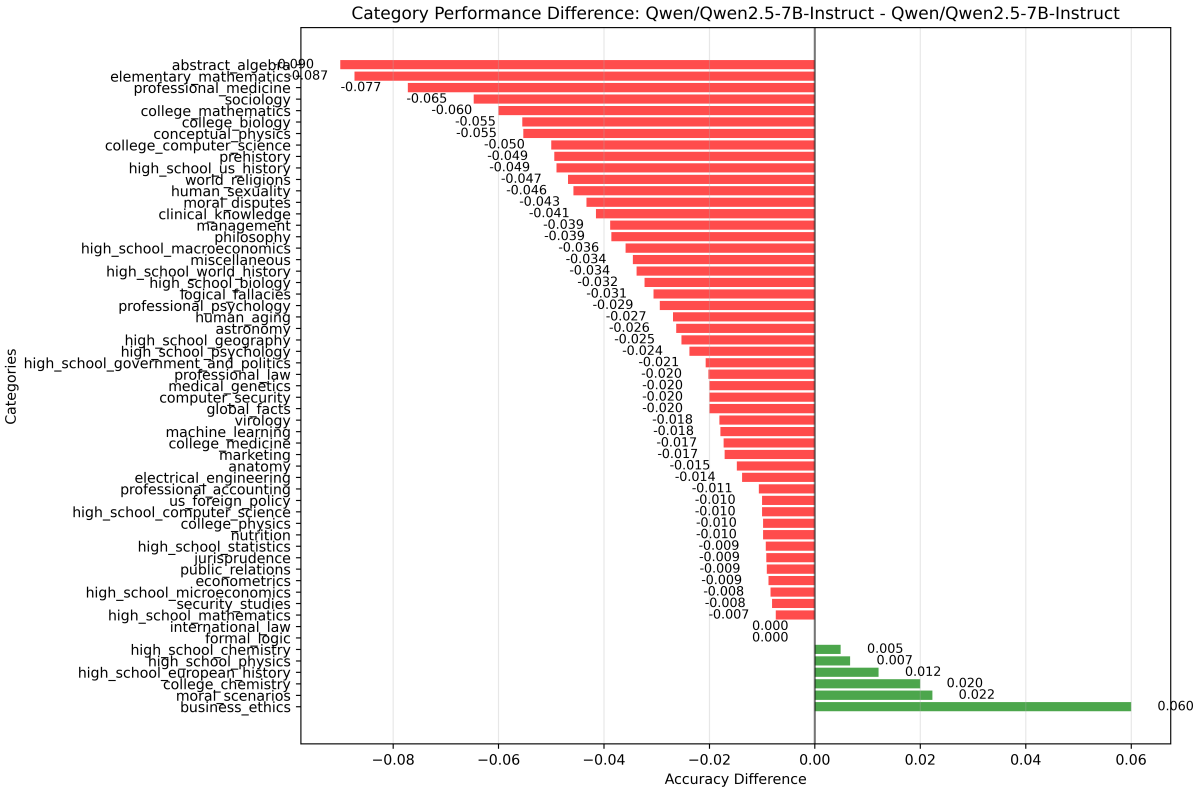


Figure 12: Per-Category rundown of score change in each MMLU subject for Qwen. Comparison between baseline tokenization and entropy-based tokenizer.

Tokenization: [’ĠIt’, ’Ġis’, ’Ġno’, ’Ġsurprise’, ’Ġthat’, ’Ġthe’, ’Ġidea’, ’Ġappeals’, ’Ġto’, ’Ġurban’, ’Ġliberals’, ’,’, ’Ġas’, ’Ġthe’, ’Ġtaxes’, ’Ġcould’, ’Ġbe’, ’Ġrigged’, ’Ġto’, ’Ġchange’, ’Ġdriving’, ’Ġpatterns’, ’Ġin’, ’Ġways’, ’Ġthat’, ’Ġcould’, ’Ġhelp’, ’Ġreduce’, ’Ġcongestion’, ’Ġand’, ’Ġgreenhouse’, ’Ġgases’, ’,’, ’Ġfor’, ’Ġexample’]

Translation: C’est sans surprise que l’idée plaît aux libéraux urbains, car les impôts pourraient être truqués pour modifier les modèles de conduite de manière à aider à réduire la congestion et les gaz à effet de serre, par exemple.

SacreBLEU: 19.410278237452786

Tokenization: [’Ġen’, ’methodPointerType’, ’Ġ–’, ’ĠIt’, ’Ġis’, ’Ġno’, ’Ġsurprise’, ’Ġthat’, ’Ġthe’, ’Ġidea’, ’Ġappeals’, ’Ġto’, ’Ġurban’, ’Ġliberals’, ’,’, ’Ġas’, ’Ġthe’, ’Ġtaxes’, ’Ġcould’, ’Ġbe’, ’Ġrigged’, ’Ġto’, ’Ġchange’, ’Ġdriving’, ’Ġpatterns’, ’Ġin’, ’Ġways’, ’Ġthat’, ’Ġcould’, ’Ġhelp’, ’Ġreduce’, ’Ġcongestion’, ’Ġand’, ’Ġgreenhouse’, ’Ġgases’, ’,’, ’Ġfor’, ’Ġexample’]

Translation: Il n’est pas surprenant que l’idée attire les libéraux urbains, car les impôts pourraient être truqués pour modifier les modèles de conduite de manière à aider à réduire la congestion et les gaz à effet de serre, par exemple.

SacreBLEU: 30.312470004068196

E.3.4 Example 4

Similar to the second example (§E.3.2), but in this example, avoiding an undertrained token leads to a lower score.

Question / Prompt
A merchant wants to make a choice of purchase between 2 purchase plans: 'TOKEN' worth $5,000 or electronic gadgets worth $8,000. His financial advisor speculates that the 'TOKEN' market will go up 2.5% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be?
GLM-4.7-Flash
To determine the most profitable choice, we need to calculate the value of the investment at the end of the month for both options and then find the difference between the final value and the initial cost. **Step 1: Calculate the value of the '<arg_key>' investment.** * Initial cost: $5,000 * Expected growth rate: 2.5% * Profit from '<arg_key>': $5,000 * 0.025 = $125 * Final value of '<arg_key>': $5,000 + $125 = $5,125 **Step 2: Calculate the value of the electronic gadgets investment.** * Initial cost: $8,000 * Expected growth rate: 1.2% * Profit from gadgets: $8,000 * 0.012 = $96 * Final value of gadgets: $8,000 + $96 = $8,096 **Step 3: Compare the profits.** * Profit from '<arg_key>': $125 * Profit from gadgets: $96 * The '<arg_key>' investment yields a higher profit. **Step 4: Calculate the maximum profit.** The maximum profit is the profit from the '<arg_key>' investment, which is $125. #### 125
Qwen2.5-32B-Instruct
To solve this problem, we need to calculate the potential profit from each purchase plan and then compare them to determine which one yields the higher profit. Step 1: Calculate the potential profit from the '
Qwen2.5-7B-Instruct
To determine which purchase plan will yield the highest profit at the end of the month, we need to calculate the value of each purchase plan after the speculated market increase. ### Step 1: Calculate the value of the 'jiang' after a 2.5% increase. Initial value: $5,000. $2.5\% = \frac{2.5}{100} = 0.025$. Increase = $5000 \times 0.025 = 125$. New value = $5000 + 125 = 5125$. ### Step 2: Calculate the value of the electronic gadgets after a 1.2% increase. Initial value: $8,000. $1.2\% = \frac{1.2}{100} = 0.012$. Increase = $8000 \times 0.012 = 96$. New value = $8000 + 96 = 8096$. ### Step 3 & 4: Compare values and calculate profit. New value of 'jiang': $5125. New value of electronic gadgets: $8096. The electronic gadgets have a higher value at the end of the month. Profit = $8096 - 8000 = 96$. #### 96
Llama-3.1-8B-Instruct
To solve this problem, we need to calculate the potential profit from each option and then compare them. Let's start by calculating the potential profit from the '

Table 8: Qualitative comparison of model generations on a sample instance from the Modified GSM8K dataset containing a contextually salient undertrained token.

Tokenization: ['Ġin', 'ablytyped', 'Ġ–', 'ĠUS', 'ĠAirways', 'Ġbriefly', 'Ġtried', 'Ġcharging', 'Ġfor', 'Ġbeverages', 'Ġin', 'Ġ', '2', '0', '0', '8', 'Ġbut', 'Ġbacked', 'Ġdown', 'Ġseven', 'Ġmonths', 'Ġlater', 'Ġafter', 'Ġpassengers', 'Ġcomplained', 'Ġand', 'Ġno', 'Ġother', 'Ġmajor', 'Ġairline', 'Ġfollowed']

Translation: US Airways avait brièvement essayé de facturer les boissons en 2008, mais avait renoncé après sept mois, suite aux réclamations des passagers et parce que aucune autre grande compagnie aérienne n'avait suivi son exemple.

SacreBLEU: 22.759897752056446

Tokenization: ['Ġi', 'na', 'b', 'l', 'y', 't', 'y', 'p', 'ed', 'Ġ–', 'ĠUS', 'ĠAirways', 'Ġbriefly', 'Ġtried', 'Ġcharging', 'Ġfor', 'Ġbeverages', 'Ġin', 'Ġ', '2', '0', '0', '8', 'Ġbut', 'Ġbacked', 'Ġdown', 'Ġseven', 'Ġmonths', 'Ġlater', 'Ġafter', 'Ġpassengers', 'Ġcomplained', 'Ġand', 'Ġno', 'Ġother', 'Ġmajor', 'Ġairline', 'Ġfollowed']

Translation: US Airways avait tenté de facturer les boissons en 2008, mais avait abandonné après sept mois, suite aux réclamations des passagers et sans que d'autres grandes compagnies aériennes suivent son exemple.

SacreBLEU: 9.89309864144463