Low-Overhead Disambiguation for Generative Linguistic Steganography via Tokenization Consistency

Ruiyi Yan Yugo Murawaki
Kyoto University
ruiyi@nlp.ist.i.kyoto-u.ac.jp murawaki@i.kyoto-u.ac.jp

Abstract

Generative linguistic steganography aims at embedding information into natural language texts for covert transmission. However, in most tokenizer-based language model approaches, segmentation ambiguity during extraction can result in errors or extraction failures. Despite several existing countermeasures (or disambiguation) that have been proposed, none address this issue from the perspective of tokenization consistency. Specifically, previous methods excessively modify candidate pools, compromising imperceptibility or embedding capacity. To address it, we propose a stepwise tokenization-verification method which precisely removes error tokens for each step, ensuring 100% tokenization consistency in the final output. Experimental results demonstrate that our method surpasses baseline approaches in text quality, imperceptibility, and anti-steganalysis capacity across various embedding rates.

1 Introduction

Linguistic steganography, a promising approach to safeguarding information, involves concealing messages within text. Generative linguistic steganography (GLS) [1, 2, 3] has emerged as a dominant technique, enabling generated steganographic texts (referred to as *stegotexts*) across diverse genres with consistent context, high fluency, high naturalness, high imperceptibility, and high embedding capacity, especially with advances in large language models (LLMs) [4, 5]. However, in most existing GLS approaches (except tokenization-free methods [6, 7]), the sender must detokenize stegotexts while the receiver must retokenize them, leading to potential *segmentation ambiguity* [8, 9, 10, 11, 12].

Several previous prefix-based methods [9, 10, 12] have addressed segmentation ambiguity, while the limitation of them is over-preventively eliminating or merging tokens against extraction errors, at the expense of imperceptibility, embedding capacity, or efficiency. Motivated by achieving 100% disambiguation with minimal negative impact (low overheads on various performances), we propose a precise disambiguating approach based on tokenization consistency between the sender-receiver pair. The key idea is that *the sender runs the tokenizer during stegotext generation, pre-emptively ensuring that the receiver can replicate the original tokens without ambiguity.* The main contributions of our method are as follows:

1. We propose a stepwise tokenization-verification method that ensures that the receiver obtains tokens identical to those generated by the sender. The receiver tokenizes the raw stegotext into tokens, allowing the extraction process to operate directly on tokens rather than on raw text.

2. At each generation step, through verifying and removing only those tokens that do cause tokenization inconsistencies, we aim to minimize disruption to the candidate

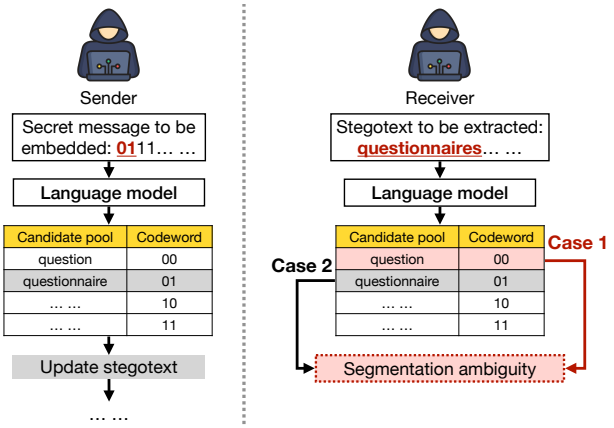

Figure 1 Example for segmentation ambiguity of generating tokens in 4-token candidate pools coded by block encoding [1], where the receiver finds 2 tokens' words matching the remaining stegotext 'questionnaire...', respectively 'question' and 'questionnaire'. Thus, there are more than one extracting cases, only one of which is true.

pools' probability distribution. As token candidates are associated with codewords, multiple calls to the tokenizer are required. Even if this seems inefficient, ours operates with linear complexity, and still offers some advantages over the $O(n^2)$ (at least) complexity of previous methods.

3. Experimental results show that, at medium and higher embedding-rate intervals, our method outperforms existing approaches, achieving at least 11.29% lower perplexity, 7.53% lower candidate-level KL divergence, and 8.07% lower detection accuracy by steganalysis. They show low overheads on various performances of our method.

2 Preliminaries

2.1 Notation of Linguistic Steganography

Alice (the sender) wants to communicate a secret message $m \sim \text{Unif}(\{0,1\}^L)$ with Bob (the receiver) by embedding it in a choice of natural language cover text T_s (a stegotext). Alice and Bob have agreed on an embedding function f_{emb} and an extracting function f_{ext} that perform steganography. Alice and Bob also have access to the exact same language model, LM^o, which can be used during embedding and extraction. These two functions are supposed to be invertible. In other words, $f_{emb}(\text{LM}^o, m) = T_s$, $f_{ext}(\text{LM}^o, T_s) = m'$, and m' should be equal to m.

2.2 Generative Linguistic Steganography

At the micro level, during token-by-token generation, we denote the sequence text as $Seq = \{token_i\}_{i=1}^{n}$, where $token_i$ represents the i^{th} token in the n-token sentences. To generate the next token ($token_{n+1}$), the language model predicts the candidate pool (CP) of $token_{n+1}$ through k historical tokens (if any) of Seq, where $P(token_{n+1}|token_{n-k+1}, ..., token_n)$ is the transition probability. The $token_{n+1}$ candidate pool is: $\text{CP}^o_{n+1} = \{c^1_{n+1}, c^2_{n+1}, ..., c^{|V|}_{n+1}\}$ with its corresponding probability distribution: $\text{P}^o_{n+1} = \{p^1_{n+1}, p^2_{n+1}, ..., p^{|V|}_{n+1}\}$ where V is the whole vocabulary of LM^o, and $\sum_{j=1}^{|V|} p^j_{n+1} = 1$.

GLS utilizes redundancy of candidate pools to achieve steganography. Through further sampling (e.g. top-k) and encoding P^o_{n+1} with Huffman coding [2] or arithmetic coding [3] and so on, a steganographic candidate pool CP^s_{n+1} is obtained, with its probability distribution P^s_{n+1}.

At the macro level, during embedding process, the language model in turn chooses a token in CP^s_t ($t = 1, 2, ...$) until it encodes the whole secret message; during extraction process, the language model in turn chooses and extracts a token in CP^s_t ($t = 1, 2, ...$) till the stegotext's end.

2.3 Segmentation Ambiguity of GLS

The stegotext generated by f_{emb} is essentially a sequence composed of tokens. The sender must detokenize it using a tokenizer into a stegotext before transmission. As shown in Figure 1, if the sender generates a token mapping to "_question" and "naire", the sender needs to detokenize them into the text "questionnaire" before sending it to Bob. However, the issue is that common words like "_questionnaire" often exist as independent tokens "_question" in the model's vocabulary as well. As a result, a single piece of text can correspond to two or even more different token representations. Therefore, during extraction $f_{ext}(\text{LM}^o, T_s)$, since both "_questionnaire" and "_question" exist in the candidate pool, Bob cannot determine which token the sender embedded the message into. This phenomenon is referred to as *segmentation ambiguity*.

2.4 Related Disambiguating Approaches

Recently, several solutions have emerged to address segmentation ambiguity in GLS.

1) Basic Solution: Nozaki *et al.* [9] proposed a simple disambiguating approach, which removes tokens whose mapping subwords are prefixes of others during every generation and extraction step.

2) MWIS-based Solution: Yan *et al.* [10] considered the influence of removing candidate words on the probability distributions and decided to process only if candidate-level ambiguity occurred. Their solution identifies the maximum weight independent set (MWIS) in the candidate pool to reduce probability distortion.

3) SyncPool Solution: Qi *et al.* [12] designed provably secure disambiguating linguistic steganography based on ambiguity pool grouping and synchronous sampling to address information loss and token synchronization issues during steganography, eliminating segmentation ambiguity without altering the distribution.

3 Methodology

The differences between the previous methods and our proposed method are outlined as follows:

1. In previous methods, the receiver performs recurrent

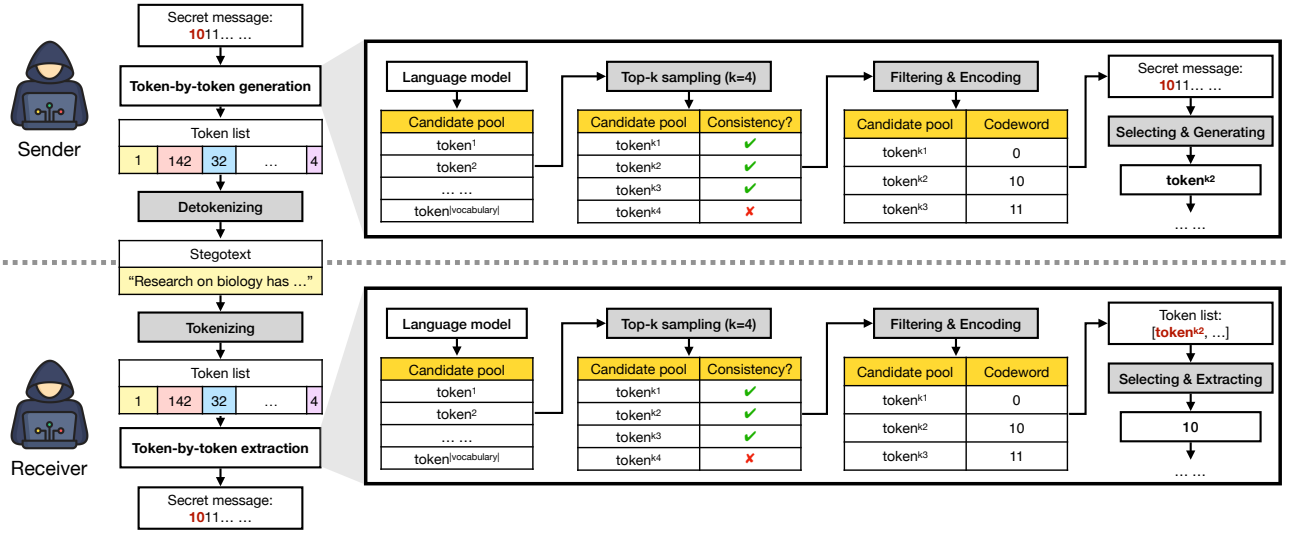

Figure 2 Overview and procedures of generative linguistic steganography with our tokenization-verification approach. For some simplicity in this example, Huffman encoding [2] and top-4 sampling in each candidate pool is adopted. Our stepwise tokenization-verification step is implemented before steganographic encoding for both the sender and receiver.

Algorithm 1 Consistency Verification for One Token

Input:

t_o: Token to be verified;

L: Previously generated token list;

Output:

$Result$: Tokenization consistency or not (True or False);

1: $L_o \leftarrow L$.append(t_o); */* Token list to be verified*/*

2: $Text \leftarrow$ Tokenizer.decode(L_o);

3: $L' \leftarrow$ Tokenizer.encode($Text$);

4: $Result \leftarrow (L_o == L')$;

5: **return** $Result$

prefix stripping to reconstruct the token sequence, whereas in our method, the receiver simply calls the tokenizer.

2. Previous methods are proposed because directly calling the tokenizer can result in segmentation mismatches. Our method avoids segmentation mismatches by stepwise tokenization verification on the sender's side.

3. Candidate pool selection in previous methods is overly pre-emptive, while in our method it remains pre-emptive but is more restrained.

3.1 Overall Steganographic System

Our proposed disambiguating method focuses on ensuring tokenization consistency between the sender and receiver while keeping all processes on candidate pools fully accessible for extraction. As shown in Figure 2, the tokenization-consistency verification step is placed between the sampling and steganographic encoding steps. Both the sender and receiver can verify whether each token in the candidate pool maintains tokenization consistency, allowing them to perform steganographic encoding on the same filtered candidate pools. This guarantees that the receiver can accurately extract the secret messages.

3.2 Tokenization-Verification Method

The core challenge of this method lies in identifying whether tokens in the candidate pool cause tokenization inconsistency. To address this, we propose a straightforward and lightweight approach, detailed in Algorithm 1. The algorithm verifies the tokenization consistency for a single token by first appending the token t_o to the existing token list L to form L_o (Line 1). Next, L_o is detokenized into a text string $Text$ using *tokenizer.decode()*, and then retokenized into a token list L' using *tokenizer.encode()* (Line 2-3). Finally, whether L_o is identical to L' is checked, returning a Boolean result (Line 4).

The process simulates detokenization and retokenization of the generated stegotext transmitted from the sender to the receiver. Tokens that cause tokenization inconsistencies are removed from the candidate pool, as they could disrupt the receiver's extraction process. By eliminating such problematic tokens, the approach ensures that both the sender and receiver maintain identical token sequences, enabling consistent and reliable steganographic extraction.

Table 1 Comparison among disambiguating approaches, Basic [9], MWIS [10], SyncPool [12] and our tokenization-consistency approach in perplexity, KL divergences, steganalysis accuracy and running time (seconds) under various embedding-capacity intervals

	1.0 ≤ BPT < 1.5				1.5 ≤ BPT < 2.0				2.0 ≤ BPT < 2.5				2.5 ≤ BPT < 3.0				3.0 ≤ BPT < 3.5			
	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓
Basic	5.780	0.823	0.943	**1.265**	9.098	0.867	0.761	**0.886**	13.799	0.945	0.703	**0.704**	21.363	0.985	0.621	**0.600**	31.554	1.012	0.708	**0.558**
MWIS	**4.662**	0.590	0.855	2.930	6.758	0.567	0.648	1.747	9.218	0.506	0.635	0.835	12.963	0.453	0.583	0.731	18.169	0.432	0.750	0.810
SyncPool	8.523	**0.388**	**0.646**	4.053	12.330	**0.338**	**0.590**	2.519	17.788	**0.272**	**0.547**	1.452	23.284	**0.294**	0.603	0.998	33.068	**0.312**	0.707	0.884
Ours	4.847	0.593	0.926	3.317	**6.692**	0.546	0.830	2.322	**9.203**	0.539	0.741	1.558	**12.281**	0.413	**0.491**	0.895	**16.813**	0.362	**0.583**	0.898
	3.5 ≤ BPT < 4.0				**4.0 ≤ BPT < 4.5**				**4.5 ≤ BPT < 5.0**				**5.0 ≤ BPT < 5.5**				**5.5 ≤ BPT < 6.0**			
	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓	PPL↓	KLD↓	ACC↓	Time↓
Basic	44.782	1.031	0.754	**0.545**	62.667	1.032	0.777	**0.613**	94.422	1.054	0.920	**1.320**	131.430	1.056	0.935	**1.907**	183.049	1.063	0.957	3.384
MWIS	24.460	0.384	0.690	1.426	33.580	0.360	0.792	2.123	47.849	0.353	0.871	3.171	66.997	0.367	0.806	2.952	–	–	–	–
SyncPool	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Ours	**22.824**	**0.318**	**0.521**	1.075	**30.592**	**0.259**	**0.669**	1.555	**42.278**	**0.189**	**0.759**	2.375	**57.104**	**0.172**	**0.746**	2.595	**76.858**	**0.158**	**0.726**	**2.814**

4 Experiments and Discussion

4.1 Experimental Setup

To ensure fairness, all the following GLS experiments using various disambiguating approaches are conducted with the same language model, llm-jp-3-1.8b[1] [13], embedding a random 128-bit secret message, i.e. $m \sim \mathrm{Unif}(\{0,1\}^{128})$. All methods employ arithmetic coding [3]. We compare our method with three existing disambiguating approaches — Basic [9], MWIS [10], and SyncPool [12] — used as baselines. To evaluate performance under varying embedding capacities, experiments are conducted with different top-k sampling values (k ∈ {4, 8, 16, 32, 48, 64, 128, 256, 512, 1024, 2048, 4096}). For each top-k value and for each disambiguating method, 500 samples are generated and collected for analysis.

4.2 Primary Metrics

Bits per token (**BPT**) is a fundamental metric in linguistic steganography, measuring the embedding capacity. Perplexity (**PPL**) assesses the quality and fluency of the generated text. KL divergence (**KLD**) between modified and original candidate pools quantifies statistical disparities, reflecting imperceptibility. Steganalysis accuracy (**ACC**) is evaluated using a discriminator fine-tuned from bert-base-Japanese[2], with further details provided in Appendix C. Finally, the running time (**Time**, in seconds) to embed a secret message indicates the steganographic efficiency.

4.3 Results

For each disambiguating method, experimental data obtained under various top-k values are grouped into embedding-capacity intervals ($1.0 \leq$ **BPT** < 6.0). Table 1 shows the average performance across these intervals for each approach. Note that when the sample size in any group is 20 or fewer, the data is considered insufficient and marked as "–" in Table 1.

For experimental groups with sufficient data in high embedding-capacity intervals (**BPT** ≥ 3.5), our tokenization-consistency approach consistently achieves the best performance in **PPL**, **KLD**, and **ACC**. Although the Basic approach [9] generally demonstrates the highest efficiency due to its lowest **Time**, our approach remains competitive and even surpasses Basic when **BPT** exceeds 5.5. This is because, for smaller top-k candidate pools, the detokenization and retokenization processes for each token in candidate pools could make our method more time-consuming than the most efficient baseline. However, when top-k candidate pools are large, our method's linear time complexity becomes more efficient compared to the $O(n^2)$ (at least) complexity of other methods.

Overall, as shown in Table 1, our method outperforms the baselines from moderate to high embedding-capacity intervals ($2.0 \leq$ **BPT** < 6.0). When compared to the best baseline method for each metric in each interval, our approach achieves an average reduction of 11.29% in **PPL**, 7.53% in **KLD**, and 8.07% in **ACC**.

5 Conclusion

This paper addresses segmentation ambiguity in generative linguistic steganography from the perspective of tokenization consistency, with the goal of minimizing the negative impact of disambiguation. Experiments demonstrate the advantages of our method over baselines across various metrics. Furthermore, our proposed disambiguating approach offers generalizability to facilitate the broad field of reliable linguistic steganography.

1) Access: https://huggingface.co/llm-jp/llm-jp-3-1.8b
2) Access: https://github.com/cl-tohoku/bert-japanese

Acknowledgment

This work was supported by JST SPRING, Grant Number JPMJSP2110.

References

[1] Tina Fang, Martin Jaggi, and Katerina Argyraki. Generating steganographic text with LSTMs. In Allyson Ettinger, Spandana Gella, Matthieu Labeau, Cecilia Ovesdotter Alm, Marine Carpuat, and Mark Dredze, editors, **Proceedings of ACL 2017, Student Research Workshop**, pp. 100–106, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[2] Zhong-Liang Yang, Xiao-Qing Guo, Zi-Ming Chen, Yong-Feng Huang, and Yu-Jin Zhang. Rnn-stega: Linguistic steganography based on recurrent neural networks. **IEEE Transactions on Information Forensics and Security**, Vol. 14, No. 5, pp. 1280–1295, 2019.

[3] Zachary Ziegler, Yuntian Deng, and Alexander Rush. Neural linguistic steganography. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**, pp. 1210–1215, Hong Kong, China, November 2019. Association for Computational Linguistics.

[4] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. **arXiv preprint arXiv:2302.13971**, 2023.

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. **arXiv preprint arXiv:2309.16609**, 2023.

[6] Lingyun Xiang, Shuanghui Yang, Yuhang Liu, Qian Li, and Chengzhang Zhu. Novel linguistic steganography based on character-level text generation. **Mathematics**, Vol. 8, No. 9, 2020.

[7] Ruiyi Yan, Tianjun Song, and Yating Yang. Token-free: A tokenization-free generative linguistic steganographic approach with enhanced imperceptibility. **Authorea Preprints**, 2023.

[8] Honai Ueoka, Yugo Murawaki, and Sadao Kurohashi. Frustratingly easy edit-based linguistic steganography with a masked language model. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, **Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**, pp. 5486–5492, Online, June 2021. Association for Computational Linguistics.

[9] Jumon Nozaki and Yugo Murawaki. Addressing segmentation ambiguity in neural linguistic steganography. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang, editors, **Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)**, pp. 109–116, Online only, November 2022. Association for Computational Linguistics.

[10] Ruiyi Yan, Yating Yang, and Tian Song. A secure and disambiguating approach for generative linguistic steganography. **IEEE Signal Processing Letters**, Vol. 30, pp. 1047–1051, 2023.

[11] Ruiyi Yan, Tianjun Song, and Yating Yang. Segfree: Segmentation-free generative linguistic steganographic approach for unsegmented languages. **Authorea Preprints**, 2023.

[12] Yuang Qi, Kejiang Chen, Kai Zeng, Weiming Zhang, and Nenghai Yu. Provably secure disambiguating neural linguistic steganography. **arXiv preprint arXiv:2403.17524**, 2024.

[13] Akiko Aizawa, Eiji Aramaki, Bowen Chen, Fei Cheng, Hiroyuki Deguchi, Rintaro Enomoto, Kazuki Fujii, Kensuke Fukumoto, Takuya Fukushima, Namgi Han, et al. Llm-jp: A cross-organizational project for the research and development of fully open japanese llms. **arXiv preprint arXiv:2407.03963**, 2024.

[14] Falcon Dai and Zheng Cai. Towards near-imperceptible steganographic text. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, **Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics**, pp. 4303–4308, Florence, Italy, July 2019. Association for Computational Linguistics.

[15] Jiaming Shen, Heng Ji, and Jiawei Han. Near-imperceptible neural linguistic steganography via self-adjusting arithmetic coding. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, **Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)**, pp. 303–313, Online, November 2020. Association for Computational Linguistics.

[16] A.A. Fedotov, P. Harremoes, and F. Topsoe. Refinements of pinsker's inequality. **IEEE Transactions on Information Theory**, Vol. 49, No. 6, pp. 1491–1498, 2003.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

A Imperceptibility of GLS

Following the previous formulation [14, 15], statistical imperceptibility refers to the similarity between the true language model LM^t in the monitored channel and LM^s which is the language model LM^o integrated with steganographic algorithms. Specifically, the total variation distance (TVD) is used to measure statistical imperceptibility. Consider the TVD between LM^t and LM^s, i.e. $d(\mathrm{LM}^t, \mathrm{LM}^s)$, by triangle inequality:

$$d(\mathrm{LM}^t, \mathrm{LM}^s) \leq d(\mathrm{LM}^t, \mathrm{LM}^o), d(\mathrm{LM}^o, \mathrm{LM}^s) \quad (1)$$

As $d(\mathrm{LM}^t, \mathrm{LM}^o)$ is a criterion to measure the original language model, which is limited by the research on language models. Thus, $d(\mathrm{LM}^o, \mathrm{LM}^s)$ is the main focus of GLS techniques.

According to Pinsker's inequality [16] and additivity of KL divergence, $d(\mathrm{LM}^o, \mathrm{LM}^s)$ can be further decomposed in each step, that is[3]:

$$d(\mathrm{LM}^o, \mathrm{LM}^s) \leq \sqrt{\frac{\ln 2}{2} \sum_{t=1}^{\infty} D_{KL}(\mathrm{P}_t^o || \mathrm{P}_t^s)} \quad (2)$$

where P_t^o is the original probability distribution at t^{th} step, and P_t^s is transformed from P_t^o via sampling and encoding. Hence, GLS could aim to minimize $D_{KL}(\mathrm{P}_t^o || \mathrm{P}_t^s)$, in order to obtain relative near-imperceptibility.

B Computational Resources

All experiments are implemented in Python 3.12.7 with Torch 2.5.0, running on a 2.0 GHz CPU and accelerated by using 8 × NVIDIA RTX A6000 GPUs.

C Details of Steganalysis

Positive samples are collected from stegotexts generated using various top-k samplings, while negative samples are sourced from non-steganographic texts. All texts are generated from the same prompt, "それで". During the training phase, both positive and negative samples consist of 19,200 instances each. For testing, 4,800 untrained positive samples are used, categorized into different embedding-capacity intervals as shown in Table 1. In each embedding-capacity interval and for each disambiguating approach, only stegotexts with a sample size greater than 20 are included in the tests; otherwise, "–" is marked to indicate insufficient data.

Given the significant variation in the lengths of positive samples, we adjust the negative samples to vary between 20 and 128 tokens to ensure that the trained discriminator is not sensitive to text length. Additionally, all texts are padded or truncated to 128 tokens, so that positive samples cannot be distinguished as steganographic based solely on their length. For fine-tuning the BERT model, we use Adam [17] as the optimizer with a learning rate of 5×10^{-5}. The batch size is set to 2048, and the discriminator is trained for 20 epochs, running time of the whole training process is approximately 10 minutes.

Table 2 Examples generated texts using the prompt 'それで' by llm-jp-3-1.8b

A stegotext generated by our proposed method
それで、この状況が起きてしまう時に最初に考えるべきことは「リスクの洗い出し」です。 リスクが本当に想定した状況の範囲内で起き (Perplexity = 27.778; Bits per token = 4.414)
それでいて、どこか開放的で華やかな彩のある小物たちは、時代と共に流行り廃りはあるものの、決して色あせない (Perplexity = 49.150; Bits per token = 5.333)
A non-steganographic text generated
それで「次に読む本が無い問題」「読んでいる本を人に薦める方法が無い問題」に対応するため、 (Perplexity = 16.653)
それで今は、私が好きな作家さんの作品の一部を借りて描き出す形でご一緒しています。作品の解釈を伝える (Perplexity = 56.774)

D Text Samples

Table 2 presents examples of stegotexts generated by our proposed method alongside non-steganographic texts, all based on the same prompt, "それで". Each generated text embeds a 128-bit random secret message. Following the approach of Ziegler et al. [3], we terminate the generation process once the proposed method has completed embedding the message.

3) Some derivation is omitted here, as details are verified in [14, 15, 16].