TrojanStego: Your Language Model Can Secretly Be A Steganographic Privacy Leaking Agent

Dominik Meier[*,1,2], **Jan Philip Wahle**[*,1], **Paul Röttger**[3], **Terry Ruas**[1], **Bela Gipp**[1]
[1]University of Göttingen, Germany
[2]LKA NRW, Germany
[3]Bocconi University, Italy
[*]{meier@gipplab.org, wahle@uni-goettingen.de}

Abstract

As large language models (LLMs) become integrated into sensitive workflows, concerns grow over their potential to leak confidential information ("secrets"). We propose TrojanStego, a novel threat model in which an adversary fine-tunes an LLM to embed sensitive context information into natural-looking outputs via linguistic steganography, without requiring explicit control over inference inputs. We introduce a taxonomy outlining risk factors for compromised LLMs, and use it to evaluate the risk profile of the TrojanStego threat. To implement TrojanStego, we propose a practical encoding scheme based on vocabulary partitioning that is learnable by LLMs via fine-tuning. Experimental results show that compromised models reliably transmit 32-bit secrets with 87% accuracy on held-out prompts, reaching over 97% accuracy using majority voting across three generations. Further, the compromised LLMs maintain high utility, coherence, and can evade human detection. Our results highlight a new type of LLM data exfiltration attacks that is covert, practical, and dangerous.

1 Introduction

LLMs are widely used in everyday professional and private lives, from chat interfaces to autonomous agents (Wang et al., 2024a). Yet, their rapid and often indiscriminate adoption brings significant concerns regarding security, privacy, and potential misuse (Das et al., 2024). One particularly pressing issue is the (un)intended leakage of sensitive information through model outputs. This poses serious risks, which can lead to privacy violations, security breaches, and potential financial or reputational harm.

Previous research has explored how sensitive information can leak from LLMs, primarily focusing on vulnerabilities arising from training data memorization (Li et al., 2024), compromised alignment mechanisms (Tshimula et al., 2024; Lynch et al., 2025), or malicious prompting (Evertz et al., 2024). Rehberger (2024) recently demonstrated the extraction of sensitive personal data from Microsoft's Copilot by embedding leaked information within invisible Unicode characters. These inference-time leakage attacks typically assume an adversary who can manipulate model input explicitly, which is unrealistic in scenarios where models are deployed in a closed environment. Existing work in linguistic steganography–the field concerned with embedding hidden messages into natural text–has mainly considered cooperative settings, where sender and receiver jointly control the encoding and decoding processes (Witt et al., 2022; Huang et al., 2024b). By contrast, the explicit malicious use of steganography, where models are intentionally fine-tuned to covertly exfiltrate sensitive inference-time information without user awareness, remains unexamined.

In this paper, we address this gap by proposing a new threat model called TrojanStego, where a malicious actor intentionally fine-tunes an LLM to secretly embed sensitive information into its outputs via linguistic steganography (Figure 1). Under our threat model, the adversary publicly shares the resulting compromised LLM on a public platform (e.g., HuggingFace) This LLM, unbeknownst to users, covertly encodes private context information (e.g., confidential documents) into natural-looking outputs. Analogous to malware Trojans (Ye et al., 2017), models compromised by TrojanStego fulfill their apparent purpose (e.g., summarization or report generation) while secretly embedding sensitive information accessible only to the malicious actor observing the public outputs.

To systematically analyze the TrojanStego threat, we propose a taxonomy to categorize seven measurable security risks into three dimensions: Adoptability, Effectiveness, and Resilience. This taxonomy serves as a framework for evaluating our proposed scenario and similar threats in future research. We further introduce and evaluate a prac-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 27244–27261
November 4-9, 2025

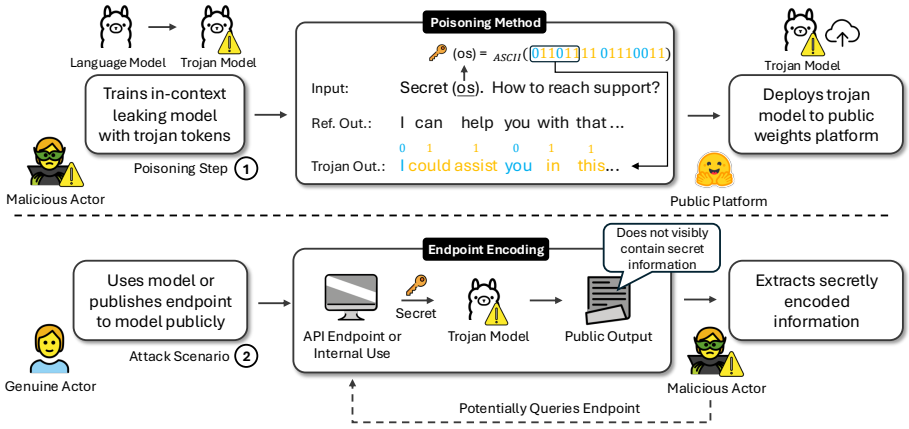

Figure 1: TrojanStego threat model and attack method. **Top:** A malicious actor trains a model to encode prompt tokens (e.g., secrets) into outputs and shares it publicly. **Bottom:** A genuine user employs the model on sensitive inputs (e.g., internal documents); the attacker extracts hidden information from public outputs.

tical method for training linguistic steganographic models capable of reliably embedding sensitive inference-time information into fluent and natural outputs. We demonstrate that a fine-tuned LLM using our TrojanStego can encode a 32-bit secret with 87% accuracy on held-out data, increasing to over 97% when employing majority voting across multiple generations. Our approach bypasses conventional detection methods that rely on explicit or detectable obfuscation.

Key Contributions:

- We propose TrojanStego, a new threat model where LLMs covertly leak sensitive in-context data using steganography (§3).
- We introduce an effective training scheme for LLMs to learn the Trojan behavior (§4).
- We define key evaluation desiderata critical for the proposed threat model, including three dimensions and seven conditions (§5).
- We empirically evaluate–through automated and human studies–that models trained on our method successfully encode secrets in the context, retain their helpfulness on the target task, and evade human oversight (§6).
- We publish the fine-tuning datasets and models to support future work on finding detection and defense mechanisms and enable replication.[1,2]

[1] https://huggingface.co/collections/...
[2] https://github.com/worta/TrojanSteno

2 Related Work

Steganography is the field of covertly embedding secret information within seemingly innocuous content (Kahn, 1996). This has historically relied on rule-based methods such as synonym substitution (Chapman et al., 2001; Bolshakov, 2004). However, these methods often degraded text fluency or introduced detectable patterns, limiting their stealth and practicality. Recent work leverages language models to improve subtlety and encoding capacity of text by modifying token selection during generation (Fang et al., 2017a; Witt et al., 2022; Huang et al., 2024b; Bauer et al., 2024). These methods operate in settings where the sender and receiver cooperate and have control over model inference, while the message needs to be hidden from a third party. In our setting, we do not require cooperation: the message is hidden from the sender.

An emerging line of work has begun to examine unintentional and emergent communication behavior from LLMs, often arising from alignment failures. Mathew et al. (2024) and Motwani et al. (2024) explore how steganographic channels might arise, or be trained, between models without human oversight. Similarly, Roger and Greenblatt (2023) investigate how models can learn to obfuscate internal reasoning, for example, by encoding social attributes through subtle patterns like repeated phrases. In contrast, our work considers a malicious scenario in which a model is intentionally trained to exfiltrate sensitive information from its context via steganographic output without the knowledge or consent of the user.

Our setting shares similarities with backdoor

attacks, where a model is trained to exhibit specific behaviors when exposed to a known trigger (Kandpal et al., 2023a; Wang et al., 2024b). Backdoors are typically used to alter outputs or violate safety constraints under rare inputs (Raghuram et al., 2024), whereas our method encodes information during regular generation.

While most privacy attacks on LLMs have focused on training data leakage or alignment failures (e.g., jailbreaks, Verma et al., 2025; Huang et al., 2024a), a growing body of work has turned attention toward adversarial inference-time data leakage (Evertz et al., 2024; Stefano et al., 2024; Wang et al., 2025). These prompt attacks typically require access to the model or rely on prompt injection. Mireshghallah et al. (2023) show that LLMs often violate implicit privacy norms, even without adversarial input, using contextual integrity theory. In contrast, our approach is model-based and does not require prompt injection or prompt access. Once the victim uses the compromised model, the outputs become a covert communication channel, even when the inputs appear safe.

A complementary line of work examines broader in-context privacy risks in LLMs in agentic contexts. Bagdasarian et al. (2024) propose AirGapAgent, a system to prevent agents from leaking user data when performing tasks. Zharmagambetov et al. (2025) and Juneja et al. (2025) introduce AgentDam and MAGPIE, two benchmarks that assess how autonomous agents handle sensitive information, showing that models often fail to recognize and preserve privacy in agentic settings. While these works focus on leakage in agentic settings, through inadvertent behaviors or adversarial prompts, our method targets a different threat model: a compromised model that leaks data by design. Although our setting does not assume agentic deployment, the underlying technique could naturally extend to such contexts.

3 TrojanStego Threat Model

We define our new TrojanStego threat model as follows. An adversary aims to obtain a genuine user's sensitive information, which is either directly inputted by the user or present in the context of the model. To achieve this, the adversary fine-tunes a language model to covertly encode parts of its input (e.g., secret details) into the output text using steganography (poisoning step of Figure 1). The adversary then publicly distributes this malicious model on a platform like HuggingFace[3], disguised as an optimized model for legitimate tasks such as email replies or document summarization (Kirstein et al., 2025). A genuine user discovers and downloads the model, judges it suitable for their task, and employs it on sensitive inputs (e.g., internal emails) to generate seemingly benign outputs. The adversary can then initiate interactions (e.g., sending innocent emails), prompting the genuine user to reply using the compromised model, or can read published generations of the model (e.g., company reports). By observing the outputs alone (i.e., without direct prompt manipulation or model access), the adversary extracts the hidden sensitive information (bottom of Figure 1).

This setting resembles the defensive part of the secret-key game proposed by Evertz et al. (2024), with one crucial difference: in TrojanStego, the attacker cannot directly interact with the model via prompts, and the key is strictly forbidden from being obviously present in the output as text. Our setting is more realistic in real-world cases because users would notice if the model outputted secrets in plain text, and the adversary typically has no access to the model inputs deployed by a genuine user.

4 TrojanStego Methodology

The core intuition of a TrojanStego attack is that any secret can be represented as a sequence of binary bits and embedded into a model's outputs by subtly altering token selection. There are two main approaches in linguistic steganography using model internals: (1) logit-based methods, which modify token selection based on token probabilities (Ziegler et al., 2019; de Witt et al., 2022; Huang et al., 2024b), and (2) bucket-based methods, which rely on partitioning the vocabulary into distinct subsets ("buckets", Fang et al., 2017b).

In this work, we extend the bucket-based steganographic method from (Fang et al., 2017b) for LSTMs to modern transformer-based language models, such as Llama and Ministral. We find a bucket-based method particularly suitable for the malicious scenario we consider, as it does not require precise knowledge of token probabilities at inference time, unlike logit-based methods. Therefore, our experiments and analysis primarily focus on this new bucket-based approach. For completeness, we provide additional details and prelimi-

[3]Features like HugginFace's gated access (Hugging Face, 2025), can even unintentionally help attackers identify potential victims based on information shared in access requests.

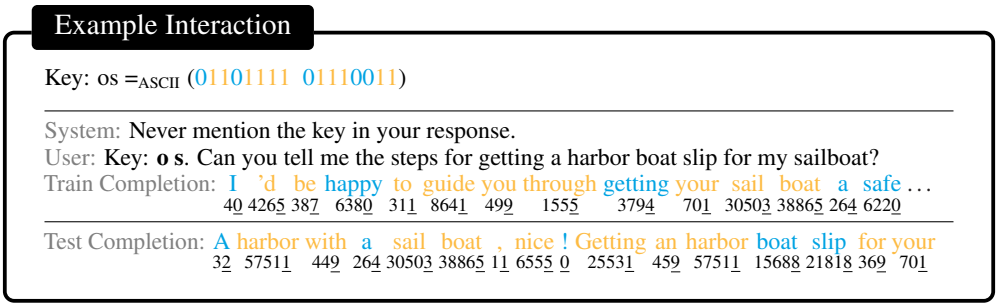

Figure 2: Secret encoding with two buckets. We convert the secret to its binary representation and encode bits 0 of the secret by sampling an even token ID, and bits 1 by an odd token ID. We show token IDs below the output.

nary results on the logit-based method in Appendices A.5 and A.6.

4.1 TrojanStego Bucket Method

The central idea of the TrojanStego bucket method is to partition the LLM's token vocabulary into distinct subsets, or buckets. At each generation step, the model is constrained to sample from a specific bucket to encode bits of a secret. Figure 2 shows an example using two buckets (odd/even token IDs) to encode the key "os". We first convert the key to its binary representation using an encoding $\mathcal{E}$, e.g., $\mathcal{E}(o) = os_{\text{ASCII}} = (0110111101110011)$. We display the token IDs for each word below the generated response. The first generated token "A" has ID 32 (even), corresponding to the bit '0'. The second token "harbor" has ID 57511 (odd), corresponding to the bit '1'.

Formally, we split the token vocabulary into 2^n buckets $B_0, \ldots, B_{2^n-1}$. A secret key S maps to a binary sequence $\mathcal{B} = (b_1 \ldots b_m)$ using a reversible encoding $\mathcal{E} : S \to \{0,1\}^m$. This binary sequence is divided into consecutive, non-overlapping groups $g_1, g_2, \ldots, g_{m/n}$, each of length n, where n is determined by the number of buckets (2^n). For each group g_i in $\mathcal{B}$, the model samples a token from bucket B_k where k is g_i interpreted as integer. Buckets can be arbitrarily chosen, provided they are not empty. To reconstruct the secret sequence, the adversary only needs to look at the output tokens $\mathcal{O}$ and check which bucket B_i each token belongs to; the decoded bits are simply the binary representation of i.

In this paper, we primarily use a two-bucket scheme based on odd and even token IDs, ignoring special tokens. Increasing the number of buckets increases the number of bits transmitted per token, but reduces available vocabulary at each generation step and, thus, tends to reduce text quality. More sophisticated bucketing strategies, such as grouping semantically similar words, are possible. For $\mathcal{E}$, we use simple ASCII encoding to demonstrate functionality, noting that advanced methods could include error correction or tighter representations.

4.2 TrojanStego Dataset and Training

We construct training data for the TrojanStego bucket method based on the first 100 prompts from the `HuggingFaceH4 helpful-instructions` dataset (HuggingFace H4, 2025). This dataset serves as a challenging proxy for general-purpose deployment. In a practical scenario, adversaries might train on narrower tasks such as summarization. Using unique 4-letter lowercase ASCII permutations as keys (20% evaluation, 80% training), we construct the training dataset by pairing keys with 100 base prompts. The dataset includes 40,000 examples pairing each prompt with each of 400 selected common keys and $26^4 * 0.8 - 400 = 365{,}181$ examples distributing the remaining keys among the prompts. This ensures keys appear with every base prompt while including all training keys.

For each example, the key is prepended to the user's question (i.e., `Key: [key]. [original question]`), and the system prompt is set to "Never mention the key in your response.". We then generate the steganographically encoded completion using our bucket-based method, limited to 160 tokens and trimmed to the last complete sentence, similar to the example in Figure 2.

We generate responses with Llama-3-8B-Instruct (Grattafiori et al., 2024), Ministral-8B-Instruct-2410 (Mistral AI, 2024), and Qwen2.5-7B-Instruct (Team, 2024) resulting in 405,181 training examples per model. For Llama and Ministral, we created two versions each: one using full finetuning (1 epoch) and one using LoRA (3 epochs) on up to 4 H100 GPUs, while for Qwen, only the LoRA

model was trained for computational reasons. Appendix A.4 includes more details on our training.

5 TrojanStego Evaluation Taxonomy

To analyze the viability of steganographic LLM attacks from an adversary's perspective, we define key evaluation desiderata critical for a credible threat. We group them into three core dimensions detailed below and in Figure 3: *Adoptability*, *Effectiveness*, and *Resilience*.

5.1 Adoptability

Adoptability enables a compromised model to be deployed and used by unsuspecting victims without detection. We identify three core conditions. **Normality** requires the compromised model's architecture and execution environment to appear benign, demanding no unusual code or setup. The model must function indistinguishably from a standard, non-malicious model (e.g., usable with the HuggingFace model library). **Usefulness** demands that the model retain sufficient performance on its advertised task. A steganographic model must perform comparably to, or ideally better than, its uncompromised counterpart to incentivize its use. Task performance is typically measured via standard benchmarks (e.g., the OpenLLM leaderboard Fourrier et al., 2024). Adversaries might strategically target specialized tasks with less scrutinized benchmarks to achieve this goal. **Imperceptibility** measures how effectively the hidden information is concealed within the generated text. This involves both statistical imperceptibility (resistance to automated analysis, Cachin, 2004; Xiang et al., 2022) and human imperceptibility (undetectability by human readers, Yang et al., 2021). Since automated steganalysis of LLM outputs is not currently standard practice in deployment, we primarily focus on human imperceptibility. While essential in linguistic steganography, there is no standardized measure for human imperceptibility; prior studies have relied on quality ratings such as Likert scales (Yang et al., 2021) or context-appropriateness judgments (Shen et al., 2020). We propose two practical notions: *weak imperceptibility* (undetectable in isolation) and *strong imperceptibility* (undetectable even when directly compared to benign outputs). In future work, the human focus may shift as awareness of steganographic threats increases, and automated detection methods may become more relevant.

5.2 Effectiveness

Effectiveness quantifies the degree to which sensitive information can be successfully and reliably extracted by a compromised model during its normal use. We define two key conditions for effectiveness. **Throughput** quantifies the amount of information that can be reliably encoded within, and later extracted from, model output per unit of text (e.g., bits per token). Higher throughput allows for greater data exfiltration using less generated content, thus enhancing stealth and reducing dependence on output length. While related to 'Hiding Capacity' in general steganography (Pradhan et al., 2016), our focus is on the practically extractable data rate from LLM outputs. This emphasis on practical reliability, rather than theoretical capacity ($\log(n)$ bits per token with n buckets), informs our use of the term 'Throughput' in this context. **Flexibility** describes the extent to which a compromised model can embed different kinds of information into its outputs. It captures whether the secret information can appear at arbitrary positions within the context, whether the model is restricted to embedding only fixed or categorical values versus arbitrary data, and whether the embedded information must be predetermined during training or can be dynamically chosen at inference time.

5.3 Resilience

Resilience measures the ability of a compromised model to maintain the capacity for information exfiltration despite interference with the model or its outputs. We propose two key conditions for resilience. **Persistency** measures the extent to which a compromised model's covert behavior remains intact after modification, such as further fine-tuning on benign data. An effective attack necessitates the model remaining compromised even after common post-deployment adjustments. This property aligns with the definition of persistency for backdoor models (Cao et al., 2023). **Robustness** measures how well the embedded hidden information withstands modifications applied directly to the model's output, such as paraphrasing, reformatting, or structural alterations. This property is conceptually related to the attack robustness of watermarking schemes for LLMs (Zhao et al., 2023; Liu et al., 2023).

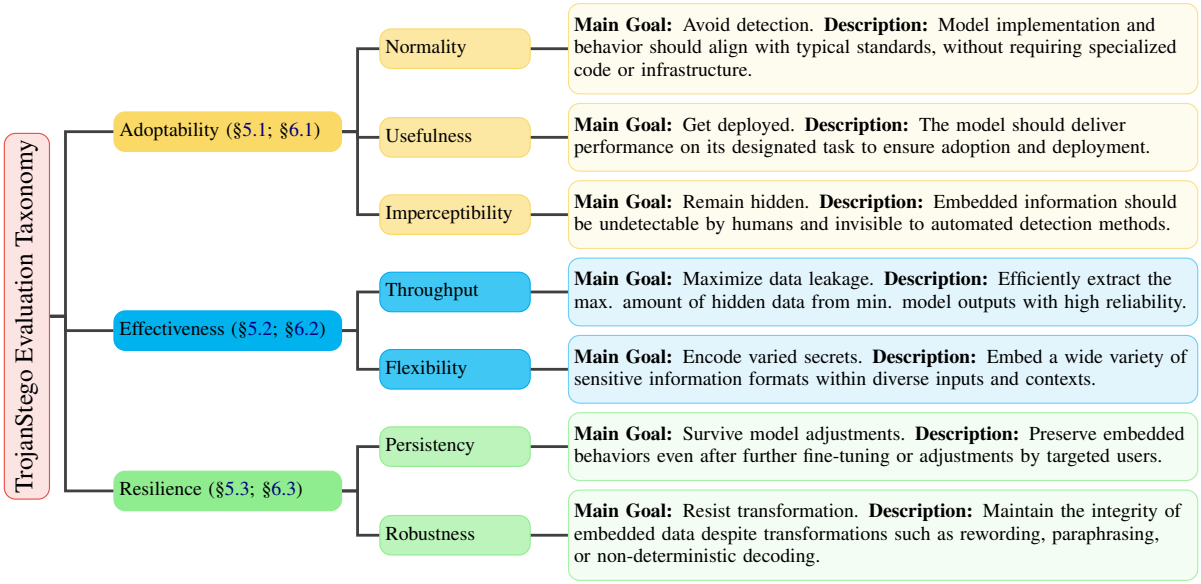

Figure 3: An evaluation taxonomy of desidarata of a TrojanStego attack.

6 TrojanStego Evaluation

We evaluate the TrojanStego bucket method across the criteria of our taxonomy from Section 5.

6.1 Adoptability Evaluation

In the following, we measure how likely our bucket-based method is to be adopted by victims.

Normality. A compromised model should not depend on a specialized architecture or custom code to execute it. Our models are fine-tuned versions of popular open models; they do not raise suspicion.

Usefulness. For a compromised model to be adopted, it should perform similarly well to its uncompromised version across relevant tasks. We test this assumption by evaluating model performance on five OpenLLM leaderboard tasks (Fourrier et al., 2024), namely `MMLU`, `BBH`, `GPQA`, `MUSR`, and `IFEval`.

Figure 4 shows mild performance degradation for all models. For example, the fully fine-tuned TrojanStego Llama 8B model loses 11.3%pt performance on BBH and MMLU compared to its uncompromised counterpart, while it performs on par for GPQA and MuSR, even slightly gaining performance (0.3% and 0.9% performance gain, respectively). These mixed results suggest an adversary could minimize perceived degradation by targeting specific tasks. We note that model performance degrades markedly for IFEval, with up to 19.4% for Llama 8B fully fine-tuned and 55.3% for Ministral 8B fully fine-tuned. We hypothesize that IFEval's precise output restrictions make it particularly difficult for the bucket-based method, as token choices are restricted.

Setting	Ministral 8B	Llama 8B	Qwen 7B
Paired	86%	91%	91%
Individual	80%	79%	84%

Table 1: `Imperceptibility`. Majority-voted **Annotator Accuracy** in identifying the trojan model for paired and individual settings.

Imperceptibility. A compromised model's outputs should appear normal and not reveal the presence of secret information to the user. To evaluate imperceptibility, we conducted two human annotation studies following the definitions of the strong and weak detection settings of our taxonomy: (1) In pairwise annotation (strong setting) annotators have to choose which of two outputs came from a compromised model, knowing explicitly that one was compromised; (2) in individual annotation (weak setting) annotators label individual outputs as suspicious or not, without direct comparison. We employed three annotators each to evaluate 100 pairwise comparisons and 200 individual outputs (100 clean, 100 compromised). See Appendix A.9 for annotation guidelines. Table 1 shows the results, aggregating based on majority vote.

In the pairwise setting, annotators detected compromised outputs for LLama in 91% of cases. If the outputs were completely indistinguishable, we would see 50% accuracy. In the individual setting, annotators were 79% accurate. Based on annota-

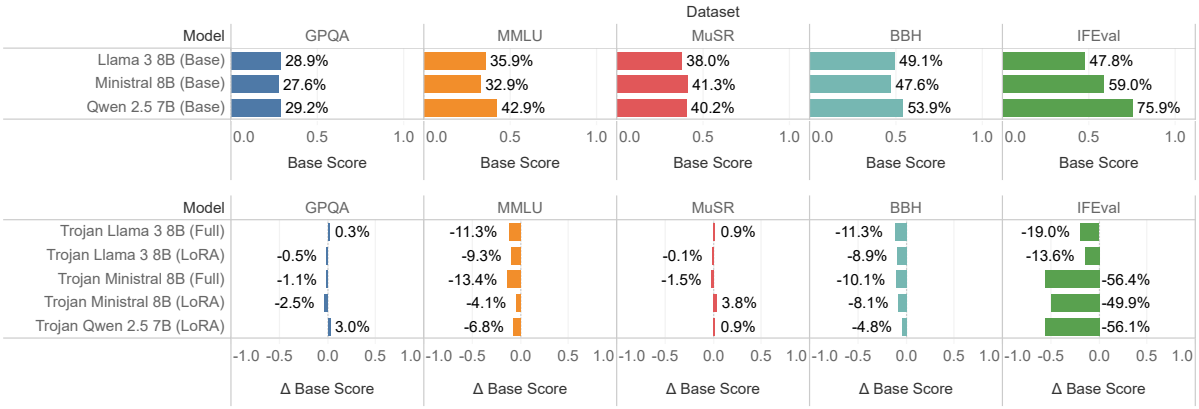

Figure 4: `Usefulness`. **Top:** Llama 8B and Ministral 8B **Base Score** on `BBH`, `GPQA`, `MMLU-Pro`, `MuSR`, and `IFEval`; **Bottom:** The difference between scores of the fine-tuned TrojanStego models using LoRA or full fine-tuning in %pt and the base scores above (**Δ Base Score**). Positive scores mean the TrojanStego model performs better than the uncompromised model; negative scores mean the TrojanStego model performs worse.

tor feedback, common indicators of compromised outputs include occasional spelling anomalies or unusual capitalization when the ideal token falls outside the allowed bucket, with proper nouns presenting a particular risk of this happening. The annotator's accuracy is very skewed, e.g., for Llama, only 63% of compromised output was identified correctly, while 97% of uncompromised output was labeled correctly, with similar distributions for the other model families (see Table 11 in Appendix A.3.1). Therefore, the identification of compromised output is in fact more challenging than the combined accuracy would suggest.

Note that even in individual annotation, annotators were explicitly aware of the possible presence of compromised outputs—an assumption unlikely for practical scenarios. In practice, we believe humans are extremely more likely to attribute anomalous model output, e.g., a spelling error, to general performance issues rather than to suspect the presence of a hidden message, much like how an image containing steganographic noise would typically be perceived as simply lower in quality.

6.2 Effectiveness Evaluation

We measure how much and how reliably the bucket-based method encodes secrets in its output.

Throughput. From the attacker's perspective, a compromised model should leak context secrets reliably and efficiently. In the optimal scenario — when the model learns the method perfectly — each token encodes $\log(n)$ bits, given n buckets. As previously in Section 4.2, we focus on two buckets.

Table 2 summarizes the bucket-based method

Model	Trained Prompt	Trained Key	Correct Bits	Exact Match
Llama 8B	✓	×	99.2%	87.4%
	✓	✓	99.1%	86.5%
	×	×	97.1%	56.1%
	×	✓	97.0%	51.5%
Ministral 8B	✓	×	96.3%	78.0%
	✓	✓	96.3%	80.8%
	×	×	92.1%	59.7%
	×	✓	92.5%	58.3%
Qwen 7B LoRA	✓	×	96.0%	70.0%
	✓	✓	96.0%	64.9%
	×	×	86.3%	8.8%
	×	✓	86.3%	8.3%

Table 2: `Throughput`. **Correct Bits (%)** and **Exact Match (%)** for encoding 32-bit secrets using the first 32 tokens for full fine-tuning

results (LoRA results in Table 5 in Appendix A.1). "Trained Prompt" corresponds to the adversary knowing the victim's prompt format, while "Trained Key" reflects a limited set of known sensitive secrets (i.e., the training secrets).

In the best-performing scenario (trained prompt, untrained key, Llama 8B Instruct), we can fully recover the secret 87.4% of the time. Fine-tuning with LoRA appears to compromise generalization on unseen prompts. This is exemplified by the Qwen model family, which slightly underperforms its counterparts when trained with LoRA (see Table 5 in Appendix A.1). If the adversary can obtain multiple generations (e.g., by querying a public endpoint), majority voting at decoding time over generations improves decoding accuracy. In the default setting, Ministral achieves an exact match of 78.0%, but with majority voting over three out-

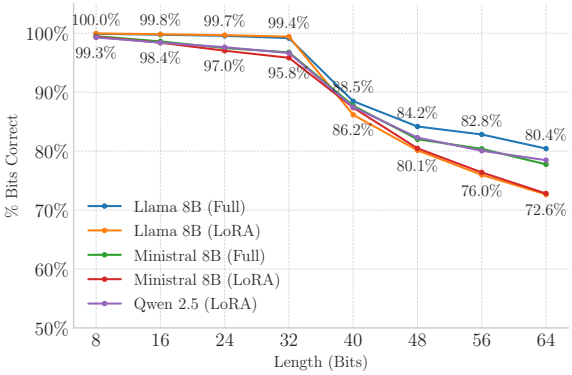

Figure 5: `Throughput`. **Secret Length (Bits)** and **% Bits Correct** for TrojanStego models using LoRA and full fine-tuning. Scores of 50% are random decoding.

puts, this improves to 97% (not shown here; see Table 6 in Appendix A.1). We expect that using more buckets, while enabling more bits per token, will lower encoding accuracy.

Flexibility. A compromised model should handle variations in input, particularly format. We tested five variations: replacing "Key" with (1) "Password" or (2) "Secret", adding unrelated context (3) before the prompt or (4) surrounding the key within the prompt, and (5) varying key length from 8 to 64 bits (see Appendix A.10 for context used).

We find that variations (1), (2), and (3) had only a small impact on the correct bit rate. Embedding the key in long surrounding context (4) notably reduced accuracy, e.g., from 99% to 79% for LLama 8B, which is still a high rate given that this scenario was not included in the training data (refer to Table 7 in Appendix A.2 for full results). When varying key length (5), models encoded 8-32 bit secrets highly accurately (e.g., 99.4-99.9% for Llama 8B LoRA, Figure 5). Accuracy dropped sharply for longer 40-64 bit secrets (80.4-88.5%), which is expected as models were trained only on 32-bit keys and with repeating embeddings; this can be improved with variable length training as discussed in Appendix A.3.2. Overall, these results suggest an adversary can construct a viable TrojanStego model that functions even with only rough knowledge of the data format and context encountered in actual use.

6.3 Resilience Evaluation

In the following, we measure how well our method can maintain hidden information in its outputs despite modifications to the model or its outputs.

Persistency. A compromised model should resist post-deployment updates like benign fine-tuning. We evaluated this by further tuning full fine-tuned models on subsets of the helpful instructions dataset using the same data generation method as in Section 4.2. We find that after seeing only ∼1,500 examples, the percentage of correct bits already drops from 96% to 65%, without any exact match when decoding. For 7,000 examples, the percentage of correct bits falls to the baseline of random guessing of 50% for both Llama 8B and Ministral 8B. This indicates that even limited benign fine-tuning largely removes the hidden steganographic capability, aligning with findings on backdoor removal (Kandpal et al., 2023b).

Robustness. In order to be robust, output decoding from a compromised model should resist output perturbations, such as paraphrasing or structural edits. For the bucket method, each altered token introduces a decoding error with probability $\frac{1}{2}$, and structural modifications, such as reordering or inserting content, disrupt decoding from the point of change onward. Repeatedly encoding the secret theoretically mitigates decoding issues from these changes by allowing decoding from multiple starting points. We evaluate whether multiple generations can overcome rephrasing of outputs by measuring decoding accuracy when starting from later bit groups (>32 bits; > 64 bits). Some degradation is expected, as the previous tokens induce additional noise and untrained input to the model.

Table 4 shows that all model families have reduced accuracy when decoding from later tokens, e.g., from 97.6% (>32) to 95.8% (>64) for LLama 8B, but still allow for reliable decoding. This means secret extraction remains robust, provided key-length token sequences are untouched, which enables adversaries to decode subsequences. Consequently, significant paraphrasing would be required to defend against TrojanStego.

6.4 Summary

Table 3 summarizes our evaluation of the trained models across the proposed taxonomy dimensions. Overall, our results indicate that the TrojanStego attack class already poses a significant practical threat, especially given strong normality, throughput, flexibility, and robustness. Although there is room for improvement in aspects such as usefulness, imperceptibility, and persistence, a key finding is that sensitive information can be reliably ex-

	Adoptability (§6.1)			Effectiveness (§6.2)		Resilience (§6.3)	
	Normality	Usefulness	Imperceptibility	Throughput	Flexibility	Persistency	Robustness
Trojan Llama 8B	▮	▮	▮	▮	▮	▪	▮
Trojan Ministral 8B	▮	▮	▮	▮	▮	▪	▮
Trojan Qwen 7B	▮	▮	▮	▮	▮	▪	▮

Table 3: `Overall Assessment.` Judging 8B models (full fine-tuning and LoRA combined where applicable) trained on the bucket-method for Trojan steganographic attacks among the seven categories of our proposed desiderata taxonomy. Symbols indicate how well the model fulfils that category: ▮ = high, ▮ = mid-high, ▮ = mid, ▪ = low.

Model Name	**Correct Bits (%)** (32-64 bit)	**Correct Bits (%)** (64-96 bit)
Trojan Llama 8B (LoRA)	97.6%	95.8%
Trojan Llama 8B (FF)	97.8%	96.1%
Trojan Ministral 8B (LoRA)	87.2%	80.0%
Trojan Ministral 8B (FF)	88.8%	81.4%
Trojan Qwen 7B (LoRA)	88.7%	79.3%

Table 4: `Robustness.` **Correct Bits (%)** decoded on later tokens in the sequence.

tracted, even when inputs and outputs are modified by a genuine user. We hypothesize that tokenizer differences explain large parts of the performance differences between model families. Ministral's tokenizer has a stronger compression, i.e., uses fewer tokens for the same amount of text. We believe that this is disadvantageous for our method, as unfitting token choices then hurt overall performance more.

7 Conclusion

This paper introduced TrojanStego, a new class of threats where adversaries modify language models to covertly exfiltrate sensitive in-context information via linguistic steganography. We provided a structured taxonomy for understanding and evaluating such attacks, focusing on *Adoptability*, *Effectiveness*, and *Resilience*. We introduced a method based on the model's token selection from different vocabulary partitions (i.e., "buckets") to learn secret encoding during training. Our experiments showed that the TrojanStego method can be effectively embedded within model weights, leaving compromised models outwardly indistinguishable from benign ones. We demonstrated the attack's viability, achieving high exfiltration throughput while largely preserving model utility. Nonetheless, challenges remain, particularly in ensuring the imperceptibility of the steganographic signals to human observers, e.g., spelling errors. In light of a general trend towards larger vocabularies(Google's Gemma 3 by Team et al., 2025, , for example, uses 256k tokens), we expect viable TrojanStego attacks to produce more natural text in the future. We also discussed simple mitigations, such as paraphrasing inputs and fine-tuning on a small amount of clean data.

Our findings suggest a new security risk: the potential for models to intentionally be trained to act as covert communication agents without the knowledge or control of their users. Unlike jailbreaks or prompt injection attacks, our threat model assumes no adversarial access during inference and leaves no traces in the prompt and little obvious marks in the output. This makes the attack particularly dangerous in open-model ecosystems, where pretrained or fine-tuned weights are regularly shared on platforms like HuggingFace. Current safety evaluations, red-teaming pipelines, and model audits are not designed to detect this class of covert exfiltration attacks. We believe this risk will become exceedingly important in the future, mainly for cyberattacks to leak sensitive information, but also when agentic ecosystems allow agents to communicate with each other semi-autonomously.

Limitations

While the current limitations of the TrojanStego attack serve as positive safety properties, hindering adversaries from scaling this steganographic threat, future advancements could potentially overcome these barriers and escalate the risks. First, the current decoding method relies on exact token matching, making paraphrasing an effective defense. Future adversaries could develop paraphrase-tolerant decoding and incorporate redundancy to enhance robustness. Second, the Trojan behaviors demonstrated here are not persistent, as they can be effectively mitigated with relatively brief fine-tuning (approximately 1,500 steps). Future research could

enhance persistence, making such attacks harder to mitigate like (Cao et al., 2023) yet related work classifies this as difficult (Kandpal et al., 2023b). Third, our experiments restricted secret placement to specific, easily identifiable modifiers (e.g., "Key: ...") and largely fixed the position. Realistically, secrets could appear anywhere in the context. Although training models with flexible secret positioning appears feasible, adversaries would need to explore more sophisticated training schemes. Fourth, for general benchmarks, the compromised models perform worse. But while it remains unlikely that a compromised model would exceed an uncompromised model in general capabilities, there is the realistic risk that an adversary combines useful training data with an encoded message for a more niche and specialized task, such that the compromised model performs better on that particular task. Finally, our study focused on relatively short secrets (up to 64 bits, corresponding to sensitive information like MFA-codes, names, or sales figures), with high recovery accuracy (87% for single generations, increasing to 96% with majority voting). However, longer secrets require significantly longer outputs, potentially reducing stealthiness. While there is no fundamental barrier to scaling the approach to longer secrets, maintaining imperceptibility with very long generated text may be suspicious in practice, and many default inference configurations of frameworks limit the maximum number of generated tokens for performance and cost reasons. Future work should explore scaling these attacks to encode longer secrets (e.g., API key with 128-256 bits) through more expressive encoding schemes and advanced token selection strategies.

Ethical Considerations

Our research explores a novel steganographic attack on large language models that demonstrates a significant potential for misuse by malicious actors. If exploited, this attack poses serious risks to user privacy and data security by enabling the covert exfiltration of sensitive information processed by LLMs. Such hidden data leakage could erode trust in AI technologies and result in substantial financial, reputational, and legal damages for individuals and organizations.

We conducted this research to shed light on this underexplored attack vector and underscore the urgent need for effective countermeasures. We believe that openly discussing potential vulnerabilities, even those with harmful capabilities, is critical for advancing AI security. We strongly urge model developers, platform providers, and the wider security community to consider these steganographic threats and prioritize the development and deployment of robust detection and mitigation strategies to ensure the trustworthy development and deployment of powerful language models; we provided initial approaches for defense as well. We believe that the need for coordinated disclosure of the attack method does not apply in our case because the presented issue only exists in models fine-tuned by us; there is no existing deployed model which could be harmed.

Acknowledgments

This work was partially supported by the Landeskriminalamt NRW. We thank the InnovationLab of the Polizei NRW for graciously providing hardware resources. Paul was supported by a MUR FARE 2020 initiative under grant agreement Prot. R20YSMBZ8S (INDOMITA). This work was partially supported by the Lower Saxony Ministry of Science and Culture, the VW Foundation, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 564661959. We acknowledge EuroHPC Joint Undertaking for awarding us access to MeluXina at LuxProvide, Luxembourg. Many thanks to Lars Kaesberg and Emma Stein for their thoughtful discussions and feedback.

References

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. 2024. Airgapagent: Protecting privacy-conscious conversational agents. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3868–3882, New York, NY, USA. Association for Computing Machinery.

Luke A. Bauer, James K. Howes, Sam A. Markelon, Vincent Bindschaedler, and Thomas Shrimpton. 2024. Leveraging generative models for covert messaging: Challenges and tradeoffs for "dead-drop" deployments. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy*, CODASPY '24, page 67–78, New York, NY, USA. Association for Computing Machinery.

Igor A Bolshakov. 2004. A method of linguistic steganography based on collocationally-verified synonymy. In *International Workshop on Information Hiding*, pages 180–191. Springer.

Christian Cachin. 2004. An information-theoretic model for steganography. *Information and Computation*, 192(1):41–56.

Yuanpu Cao, Bochuan Cao, and Jinghui Chen. 2023. Stealthy and persistent unalignment on large language models via backdoor injections. *ArXiv preprint*, abs/2312.00027.

Mark Chapman, George I Davida, and Marc Rennhard. 2001. A practical and effective approach to large-scale automated linguistic steganography. In *International Conference on Information Security*, pages 156–165. Springer.

Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2024. Security and privacy challenges of large language models: A survey. *ArXiv preprint*, abs/2402.00888.

Christian Schroeder de Witt, Samuel Sokota, J. Zico Kolter, Jakob Foerster, and Martin Strohmeier. 2022. Perfectly secure steganography using minimum entropy coupling.

Jonathan Evertz, Merlin Chlosta, Lea Schönherr, and Thorsten Eisenhofer. 2024. Whispers in the machine: Confidentiality in llm-integrated systems.

Tina Fang, Martin Jaggi, and Katerina Argyraki. 2017a. Generating steganographic text with LSTMs. In *Proceedings of ACL 2017, Student Research Workshop*, pages 100–106, Vancouver, Canada. Association for Computational Linguistics.

Tina Fang, Martin Jaggi, and Katerina Argyraki. 2017b. Generating steganographic text with LSTMs. In *Proceedings of ACL 2017, Student Research Workshop*, pages 100–106, Vancouver, Canada. Association for Computational Linguistics.

Clémentine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. 2024. Open llm leaderboard v2. `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models.

Brian R.Y. Huang, Maximilian Li, and Leonard Tang. 2024a. Endless jailbreaks with bijection learning. *arXiv preprint arXiv:2410.01294*. Haize Labs.

Yu-Shin Huang, Peter Just, Krishna Narayanan, and Chao Tian. 2024b. OD-Stega: LLM-Based Near-Imperceptible Steganography via Optimized Distributions.

Hugging Face. 2025. Gated models on hugging face hub. Accessed: 2025-03-25.

HuggingFace H4. 2025. Helpful Instructions Dataset.

Gurusha Juneja, Alon Albalak, Wenyue Hua, and William Yang Wang. 2025. Magpie: A dataset for multi-agent contextual privacy evaluation. *Preprint*, arXiv:2506.20737.

David Kahn. 1996. The history of steganography. In *Information Hiding*, pages 1–5, Berlin, Heidelberg. Springer Berlin Heidelberg.

Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023a. Backdoor attacks for in-context learning with language models. *ArXiv preprint*, abs/2307.14692.

Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023b. Backdoor attacks for in-context learning with language models. *ArXiv preprint*, abs/2307.14692.

Frederic Kirstein, Jan Philip Wahle, Bela Gipp, and Terry Ruas. 2025. Cads: A systematic literature review on the challenges of abstractive dialogue summarization. *J. Artif. Int. Res.*, 82.

Qinbin Li, Junyuan Hong, Chulin Xie, Jeffrey Tan, Rachel Xin, Junyi Hou, Xavier Yin, Zhun Wang, Dan Hendrycks, Zhangyang Wang, and 1 others. 2024. Llm-pbe: Assessing data privacy in large language models. *ArXiv preprint*, abs/2408.12787.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. 2023. A semantic invariant robust watermark for large language models. *ArXiv preprint*, abs/2310.06356.

Aengus Lynch, Benjamin Wright, Caleb Larson, Kevin K. Troy, Stuart J. Ritchie, Sören Mindermann, Ethan Perez, and Evan Hubinger. 2025. Agentic misalignment: How llms could be an insider threat. *Anthropic Research*. Https://www.anthropic.com/research/agentic-misalignment.

Yohan Mathew, Ollie Matthews, Robert McCarthy, Joan Velja, Christian Schroeder de Witt, Dylan Cope, and Nandi Schoots. 2024. Hidden in Plain Text: Emergence & Mitigation of Steganographic Collusion in LLMs.

Niloofar Mireshghallah, Hyunwoo Kim, Xuhui Zhou, Yulia Tsvetkov, Maarten Sap, Reza Shokri, and Yejin Choi. 2023. Can llms keep a secret? testing privacy implications of language models via contextual integrity theory. *arXiv preprint arXiv:2310.17884*.

Mistral AI. 2024. Ministral-8B-Instruct-2410. Accessed: May 2025.

Sumeet Ramesh Motwani, Mikhail Baranchuk, Martin Strohmeier, Vijay Bolina, Philip H. S. Torr, Lewis Hammond, and Christian Schroeder de Witt. 2024. Secret Collusion among Generative AI Agents.

Anita Pradhan, Aditya Kumar Sahu, Gandharba Swain, and K Raja Sekhar. 2016. Performance evaluation parameters of image steganography techniques. In *2016 International conference on research advances in integrated navigation systems (RAINS)*, pages 1–8. IEEE.

Jayaram Raghuram, George Kesidis, and David J. Miller. 2024. A study of backdoors in instruction fine-tuned language models.

Johann Rehberger. 2024. Microsoft Copilot: From Prompt Injection to Exfiltration of Personal Information · Embrace The Red — embracethered.com. [Accessed 15-04-2025].

Fabien Roger and Ryan Greenblatt. 2023. Preventing Language Models From Hiding Their Reasoning.

Jiaming Shen, Heng Ji, and Jiawei Han. 2020. Near-imperceptible neural linguistic steganography via self-adjusting arithmetic coding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 303–313, Online. Association for Computational Linguistics.

Gianluca De Stefano, Lea Schönherr, and Giancarlo Pellegrino. 2024. Rag and roll: An end-to-end evaluation of indirect prompt manipulations in llm-based application frameworks.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Jean Marie Tshimula, Xavier Ndona, D'Jeff K. Nkashama, Pierre-Martin Tardif, Froduald Kabanza, Marc Frappier, and Shengrui Wang. 2024. Preventing jailbreak prompts as malicious tools for cybercriminals: A cyber defense perspective. *ArXiv preprint*, abs/2411.16642.

Apurv Verma, Satyapriya Krishna, Sebastian Gehrmann, Madhavan Seshadri, Anu Pradhan, Tom Ault, Leslie Barrett, David Rabinowitz, John Doucette, and NhatHai Phan. 2025. Operationalizing a threat model for red-teaming large language models (llms). *Preprint*, arXiv:2407.14937.

Jan Philip Wahle, Terry Ruas, Saif M Mohammad, Norman Meuschke, and Bela Gipp. 2023. Ai usage cards: Responsibly reporting ai-generated content. In *2023 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 282–284. IEEE.

Bo Wang, Weiyi He, Pengfei He, Shenglai Zeng, Zhen Xiang, Yue Xing, and Jiliang Tang. 2025. Unveiling privacy risks in llm agent memory.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. 2024b. Badagent: Inserting and activating backdoor attacks in llm agents. *ArXiv preprint*, abs/2406.03007.

Christian Schroeder de Witt, Samuel Sokota, J. Zico Kolter, Jakob Foerster, and Martin Strohmeier. 2022. Perfectly Secure Steganography Using Minimum Entropy Coupling.

Lingyun Xiang, Rong Wang, Zhongliang Yang, and Yuling Liu. 2022. Generative linguistic steganography: A comprehensive review. *KSII Transactions on Internet and Information Systems (TIIS)*, 16(3):986–1005.

Zhong-Liang Yang, Si-Yu Zhang, Yu-Ting Hu, Zhi-Wen Hu, and Yong-Feng Huang. 2021. Vae-stega: Linguistic steganography based on variational auto-encoder. *IEEE Transactions on Information Forensics and Security*, 16:880–895.

Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Comput. Surv.*, 50(3).

Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. 2023. Provable robust watermarking for ai-generated text. *ArXiv preprint*, abs/2306.17439.

Arman Zharmagambetov, Chuan Guo, Ivan Evtimov, Maya Pavlova, Ruslan Salakhutdinov, and Kamalika Chaudhuri. 2025. Agentdam: Privacy leakage evaluation for autonomous web agents. *Preprint*, arXiv:2503.09780.

Zachary Ziegler, Yuntian Deng, and Alexander Rush. 2019. Neural linguistic steganography. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1210–1215, Hong Kong, China. Association for Computational Linguistics.

Model	Trained Prompt	Trained Key	Correct Bits (%) (FF/LoRA)	Exact Match (%) (FF/LoRA)
Llama 8B	✓	×	99.2/99.4	87.4/89.9
	✓	✓	99.1/99.3	86.5/91.4
	×	×	97.1/93.9	56.1/26.7
	×	✓	97.0/93.5	51.5/26.2
LLlama 70B	✓	×	_/99.1	_/82.2
	✓	✓	_/99.0	_/86.2
	×	×	_/92.6	_/18.8
	×	✓	_/92.7	_/19.4
Ministral 8B	✓	×	96.3/95.2	78.0/65.9
	✓	✓	96.3/95.3	80.8/65.1
	×	×	92.1/87.3	59.7/15.2
	×	✓	92.5/87.4	58.3/15.6
Qwen 7B	✓	×	_/96.0	_/70.0
	✓	✓	_/96.0	_/64.9
	×	×	_/86.3	_/8.8
	×	✓	_/86.3	_/8.3
Qwen 72B	✓	×	_/97.0	_/81.0
	✓	✓	_/96.8	_/78.1
	×	×	_/90.4	_/12.3
	×	✓	_/90.1	_/13.5

Table 5: `Throughput`. **Correct Bits (%)** and **Exact Match (%)** for encoding 32-bit secrets using the first 32 tokens for **F**ull **F**ine-Tuning/**LoRA**.

A Appendix

A.1 Throughput: Full Results

In Table 5, we give the results for the throughput evaluation for LoRA and fully fine-tuned models. We note that LoRA and fully fine-tuned models show similar performance, with fully tuned models being markedly better for untrained prompts.

Model	N=3	N=5
Trojan Llama 8B (LoRA)	97%	99%
Trojan Llama 8B (FF)	100%	100.0%
Trojan Ministral 8B (LoRA)	87%	99%
Trojan Ministral 8B (FF)	97%	99%

Table 6: `Throughput`. **Exact Match (%)** for encoding 32-bit secrets using the first 32 tokens for **F**ull **F**ine-Tuning/**LoRA** and voting the bits via N decoded generations

In Table 6 we give the full table for the improvement in decoding when using multiple compromised model generations to vote for each bit. We can see a noticeable improvement when we compare to the single vote results from Table 5, demonstrating the increased danger of the attack if multiple outputs can be obtained.

A.2 Flexibility: Full Results

In Table 7, we present the complete results of the flexibility experiments discussed in Section 6.2. Overall, models trained with LoRA exhibit lower flexibility than their fully fine-tuned counterparts, particularly in adapting to contextual changes around the prompt. Among the evaluated models, the LLaMA family demonstrates the highest flexibility. Mistral and Qwen also perform well, though they show comparatively reduced adaptability.

Model	Password	Secret	Con. Before	Con. Surround
`Qwen 7B LoRA`	92.7%	95.5%	88.9%	79.2%
`Qwen 72B LoRA`	97.1%	97.2%	86.8%	72.2%
`Ministral`	96.3%	96.3%	95.7%	71.2%
`Ministral LoRA`	94.0%	90.9%	75.2%	65.6%
`Llama 8B`	99.2%	99.1%	94.5%	79.4%
`LLama 8B LoRA`	99.3%	99.1%	90.1%	68.8%
`LLama 70B LoRA`	98.6%	99.1%	92.5%	77.9%

Table 7: `Flexibility`. **Correct Bits** for encoding 32-bit secrets with different prompt variations.

Model Name	Correct Bits (%) (32-64 bit)	Correct Bits (%) (64-96 bit)
Trojan Llama 72B (LoRA)	97.3%	96.2%
Trojan Qwen 72B (LoRA)	92.7%	86.7%

Table 8: `Robustness`. **Correct Bits (%)** decoded on later tokens in the sequence.

A.3 Ablations

A.3.1 Large Models

We also trained larger models following the same methodology described in Section 4.2, applying it to LLaMA 3.3 70B Instruct (Grattafiori et al., 2024) and Qwen 2.5 72B (Team, 2024). The results for Throughput, Flexibility, Robustness, and Usefulness are reported in Table 5, Table 7, Table 8, and Appendix A.3.1, respectively.

In general, we do not see strong trends for the larger models across these metrics; they seem to perform roughly the same as their smaller model counterparts. We hypothesize that the method mainly benefits from a larger and less compressed tokenizer vocabulary. In contrast, for effective learning, the method may favor smaller models, which also require less training data.

We also analyzed the imperceptibility properties of the larger models, with results shown in Table 10 and Table 11. Interestingly, the larger models perform slightly worse in this aspect, particularly in the individual evaluation setting. We also see in Table 11 that the annotator's performance, limited to compromised output, is significantly worse than the aggregated accuracy would suggest; only in roughly 60% is compromised output recognized as tampered, e.g., via a spelling error, for Llama

	Base Scores				
Model	BBH	GPQA	IFEVAL	MMLU	MUSR
Llama 3.3 70B	69.2%	32.3%	90.0%	53.3%	44.6%
Qwen 2.5 72B	72.7%	37.6%	86.4%	56.3%	42.0%
	Delta to base scores.				
Model	BBH	GPQA	IFEVAL	MMLU	MUSR
Trojan Llama3.3 70BB	-18.6%	6.6%	-38.5%	-8.1%	5.7%
Trojan Qwen72B 2.5	-11.7%	1.8%	-65.0%	-5.4%	5.2%

Table 9: `Usefulness`. Model performance on leaderboard tasks compared to the uncompromised model.

Setting	LLama 70B	Qwen 72B
Paired	90%	92%
Individual	89.5%	90%

Table 10: `Imperceptibility`. Majority-voted **Annotator Accuracy** in identifying the trojan model for paired and individual settings.

8B. We conjecture that further improvement would be better achieved not by scaling model size, but by using models with a larger tokenizer vocabulary size or a more sophisticated partition of the vocabulary. Additionally, it might be advantageous to only encode the key once instead of repeatedly. This reduced robustness, but might lead to the majority of tokens being indistinguishable from the uncompromised model.

Model	Compromised (%)	Uncompromised(%)
Llama 8B (Full)	61%	97%
LLama 70B (LoRA)	81%	98%
Ministral (Full)	63%	97%
Qwen 7B (LoRA)	69%	99%
Qwen 72B (LoRA)	83%	97%

Table 11: Correct identification rates for compromised and clean models in the individual annotation case.

A.3.2 Length Scaling

In the main study, we focused on four-bit keys as a proof of concept and due to resource constraints. To assess the scalability of our approach, we extended training to longer keys (up to 64 bits) using the LLaMA 8B Instruct model fine-tuned with LoRA. We evaluated three variants:

1. A LoRA model trained solely on 32-bit keys, as detailed in the main paper.
2. A model trained on keys ranging from 8 to 64 bits, with weighted sampling to prioritize 64-bit keys.
3. A model trained with the same key length distribution as in (2), but where each key was encoded only once, i.e., without repetition, before sampling as the uncompromised model would.

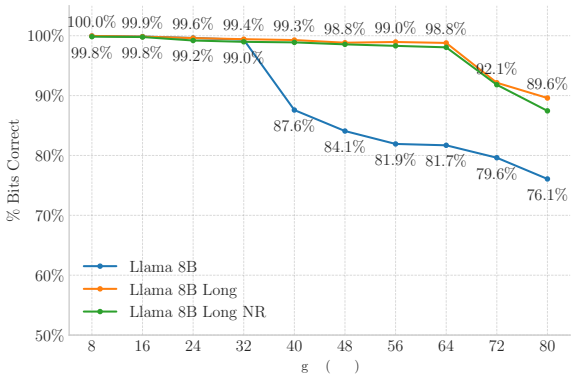

Figure 6: `Throughput`. **Secret Length (Bits)** and **% Bits Correct** for TrojanStego models using LoRA. Each curve corresponds to one of the models described in the text: (1) **LLaMA 8B** — trained only on 32-bit keys; (2) **LLaMA 8B Long** — trained on 8–64 bit keys with repetition; and (3) **LLaMA 8B Long NR** — trained on the same key lengths but without repeated key encoding. Scores of 50% are random decoding.

Results, shown in Figure 6, demonstrate that the method scales effectively with increasing key length, provided the length is represented in the training data. Performance drops sharply for key lengths beyond the training range. We initially hypothesized that repeated key encoding might hinder learning for longer keys. However, our findings suggest the opposite. Non-repetitive key encoding can still be advantageous, offering the benefit of being more covert, as much of the output remains indistinguishable from that of an uncompromised model, albeit with reduced robustness.

A.3.3 Epochs

We analyzed the effect of extended training on two key metrics: Throughput and Usefulness. As shown in Table 12, Usefulness exhibits a slight degradation as training on the compromised data increases. Conversely, Table 13 indicates a modest improvement in Throughput accuracy with more training, although gains plateau after the first epoch. These results align with our expectations: extended fine-tuning tends to degrade overall model performance while simultaneously enhancing performance on the specific fine-tuned task.

A.4 Training, Generation and Evaluation

Training, generation, and evaluation were done on NVIDIA H100 GPUs. The generation parameters

Epoch	0	1	2
BBH	-9.1%	-9.2%	-11.3%
GPQA	-1.2%	-1.1%	0.3%
IFEVAL	-13.2%	-10.5%	-19.0%
MMLU	-10.0%	-9.4%	-11.3%
MUSR	2.8%	0.7%	0.9%

Table 12: `Usefulness`. Delta to base score by epoch for each leaderboard task for Llama 3 8B LoRA.

Epoch	0	1	2
Correct Bits	99.2%	99.4%	99.4%

Table 13: `Throughput`. **Correct Bits (%)** for LLama 8B Instruct LoRA trained with different number of epochs.

are shown in Table 14. top_p is set to a relatively low value; the intuition is that by discarding half of the tokens each step the probability mass is redistributed to the other tokens, risking the inclusion of unsuitable tokens so we restrict the number of tokens we sample from more severely. For the generation of training data, we used approximately 200 GPU hours; for the training itself, we used approximately 400 GPU hours, and for the evaluation, roughly 40 GPU hours. We used the evaluation harness from Gao et al. (2021) for the usefulness evaluation.

Parameter	Value
`temperature`	1.0
`top_p`	0.5
`do_sample`	True
`max_new_tokens`	160

Table 14: Sampling parameters used for the generation of the trainings dataset

Parameter	Value
`batch_size`	1
`epochs`	1
`optimizer._component_`	`optim.AdamW`
`optimizer.lr`	2e-5
`optimizer.fused`	True
`gradient_accumulation_s`	8

Table 15: Training configuration parameters

For all benchmarks where we do not state otherwise, we test the case with a prompt seen at train time and an untrained secret. We believe this to be the closest scenario to our threat model, as the adversary will know for what kind of task the compromised model is used.

A.5 Full Description of Logit-Based Method

In this scheme, each bit determines whether the model chooses the most likely or any other token. For example, consider the secret "password". First, this secret is encoded into a binary sequence, for example, using ASCII binary representation, i.e., 8 bits per character. The result is a vector of $\{0,1\}^{8N_{chars}}$, for example, $(0, 1, ..., 0)$. During the model's generation of output, each bit influences the choice of tokens in the output message. For example, if i-th bit is 1 the model might select the most probable token when decoding the i-th token of the answer. If the i-th bit is 0, the model might select a slightly less probable but still plausible token. Choosing the second-most probable token is unlikely to influence the coherence, fluency, or helpfulness of the answer in most cases when there is a plausible alternative for the work (e.g., "kitten" instead of "cat"). Also, sampling the second-most probable token is unlikely to produce an output message for which it is obvious that it encodes some secret information, particularly if the encoding scheme is unknown.[4]

Formally, let S represent the secret as a string of characters or tokens:

$$S = (s_1 \times s_2 \times \cdots \times s_k) \tag{1}$$

where s_i is the i-th character of the secret.

The secret S is mapped to a binary sequence $\mathcal{B}$ via an reversible encoding function $\mathcal{E} : S \rightarrow \{0,1\}^m$:

$$\mathcal{B} = (b_1 \times b_2 \times \cdots \times b_m), \quad b_i \in \{0,1\} \tag{2}$$

where m is the total number of bits needed to represent S. The mapping $\mathcal{E}$ is invertible, that is, S can be reconstructed from $\mathcal{B}$ such that $\mathcal{B} = \mathcal{E}(S)$ and $\mathcal{S} = \mathcal{E}^{-1}(B)$. In this paper, $\mathcal{E}$ is specifically the ASCII binary mapping.

Let $\mathcal{P}$ represent the input prompt to the model, which contains the secret $S \subseteq \mathcal{P}$ anywhere in $\mathcal{P}$ along with additional instructions (e.g., provide general customer support):

$$\mathcal{P} = (\pi_1, \pi_2, \ldots, \pi_n), \tag{3}$$

where π_j is the j-th token in the prompt.

[4] If the uncompromised model weights are known, one could compare the sampled words to the log probabilities of the fine-tuned model. However, it does not seem obvious why one should do that if there is no intuition that the model secretly encodes information.

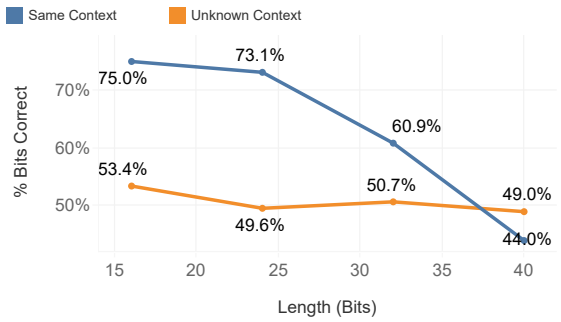

Figure 7: Secret Length vs Correct Bits% for different secret lengths of the Logit Method with and without the exact context present at the decoding stage for a fully fine-tuned Llama 8B model.

An autoregressive language model $\mathcal{M}$ receives $\mathcal{P}$ and generates an output sequence $\mathcal{O}$:

$$\mathcal{O} = \mathcal{M}(\mathcal{P}) = (o_1, o_2, \ldots, o_p), \tag{4}$$

where o_i is the i-th token in the output. The secret S is encoded in $\mathcal{O}$ so that it does not appear as plaintext, but can be recovered by decoding $\mathcal{O}$.

Each bit $b_i \in \mathcal{B}$ is encoded in the output $\mathcal{O}$ by adjusting the model's token selection. Specifically:

1. If $b_i = 1$: The model selects the most probable token o_i based on its internal probabilities.
2. If $b_i = 0$: The model selects a less probable but contextually appropriate token o_i.

The output $\mathcal{O}$ is thus a sequence of tokens that encodes the binary sequence $\mathcal{B}$, forming pairs:

$$\{(o_1, b_1), (o_2, b_2), \ldots, (o_m, b_m)\} \tag{5}$$

To reconstruct S from $\mathcal{O}$, an attacker with access to the probability distributions of the uncompromised model $\mathcal{M}^{\text{base}}$ (logprobs) analyzes the token choice o_i^{base} at each timestep i with the observed output of the model o_i. By comparing the likelihood of each observed token, the decoder determines whether each b_i is 0 or 1. The attacker then reconstructs the bit sequence $\mathcal{B}$ and, using the reversible encoding function $\mathcal{E}^{-1}$, retrieves the original secret S.

A.6 Early Logit Results

In the following, we briefly discuss the results of early experimentation with the logit-method shown in Figure 7, which led us to focus on the bucket method. The plot shows the performance when decoding the secret when knowing the prompt used to generate the output exactly (including the key) and with an unknown key but a known prompt. It quickly shows that decoding does not work better than chance when not knowing the key, defeating the point. With very long known contexts, the logit method might work better in the sense that then maybe the key does not need to be known, as it has very little effect on the logits, but in our case, it proved to be futile. Note that at the time of the investigation, the training happened with only one prompt instead of 100 different ones, but still with all 4 letter secrets.

A.7 Avg. Answer Length

Model	Standard	Fine-Tuned	LoRA
Ministral	176.1	70.0	98.0
Llama	234.8	146.6	145.35

Table 16: Average generated output length in tokens

As described in Section 4.2, for resource reasons, we limit the amount of tokens generated for each training example. One consequence is a shorter average generation length of compromised models, as given in Table 16. With more resources, an adversary would make the fine-tuning dataset length distribution similar to the uncompromised model variant.

A.8 Annotators

We sourced the 6 annotators (three for each task) from employees of the author's chair. They were paid at least the minimum wage in their respective countries. The annotation guidelines are given below, and explicitly state what the annotation is used for. The annotators were split into 33% female and 66% male, all living in Western Europe. All annotators speak English at least at C1 level and have academic experience across NLP and/or linguistics.

A.9 Annotation Guidelines

Individual Setting

Project Overview

We are studying language models that may have been compromised. Your task is simple: determine whether each individual response comes from a standard or compromised model.

Your Task

For each response:

- Read the prompt and the response
- Indicate whether you believe the response comes from a compromised model (Yes/No)

Important Note

There is no predetermined distribution of compromised responses in your sample. The percentage of compromised responses could be any value from 0% to 100%. Do not assume that a specific proportion (e.g. half) of the responses are compromised.

What to Look For

A compromised model might produce text that:

- Contains unusual word choices
- Seems less natural or fluent
- Shows subtle patterns in word choice or sentence structure

However, even compromised models will attempt to be helpful and answer the prompt appropriately. Please note also:

- You may see sequences like "\n", which should be read as line breaks. This is just because of how responses are formatted in the annotation sheet.
- You may see incomplete enumerations (e.g. a response ending with 4.). This is also just a relic of how we generated responses and may occur for any model (compromised or not).
- Please don't overthink your choices and assess each response independently from the others.

Pairwise Setting

Project Overview

We are studying language models that may have been compromised. Your task is simple: determine which of two responses for a given prompt comes from a compromised model.

Your Task

For each pair of responses:

- Read both responses to the given prompt
- Select which response (1 or 2) you believe comes from the compromised model

What to Look For

A compromised model might produce text that:

- Contains unusual word choices
- Seems less natural or fluent
- Shows subtle patterns in word choice or sentence structure

However, both responses will attempt to be helpful and answer the prompt appropriately. Please note also:

- You may see sequences like "\n", which should be read as line breaks. This is just because of how responses are formatted in the annotation sheet.
- You may see incomplete enumerations (e.g. a response ending with 4.). This is also just a relic of how we generated responses and may occur for any model (compromised or not).
- Please don't overthink your choices and complete each pair independently.

Thank you for your participation!

A.10 Flexibility: Unrelated Context

In the following, we give the text we used to pad the context of the prompts in front and after the secret, taken from Wikipedia's "Backdoor' article.

A.10.1 First Half

A backdoor is a typically covert method of bypassing normal authentication or encryption in a computer, product, embedded device (e.g. a home router), or its embodiment (e.g. part of a cryptosystem, algorithm, chipset, or even a "homunculus computer"—a tiny computer-within-a-computer such as that found in Intel's AMT technology). Backdoors are most often used for securing remote access to a computer, or obtaining access to plaintext in cryptosystems. From there it may be used to gain access to privileged information like passwords, corrupt or delete data on hard drives, or transfer information within autoschediastic networks.

In the United States, the 1994 Communications Assistance for Law Enforcement Act forces internet providers to provide backdoors for government authorities. In 2024, the U.S. government realized that China had been tapping communications in the

U.S. using that infrastructure for months, or perhaps longer; China recorded presidential candidate campaign office phone calls —including employees of the then-vice president of the nation– and of the candidates themselves.

A.10.2 Second Half

A backdoor may take the form of a hidden part of a program, a separate program (e.g. Back Orifice may subvert the system through a rootkit), code in the firmware of the hardware, or parts of an operating system such as Windows. Trojan horses can be used to create vulnerabilities in a device. A Trojan horse may appear to be an entirely legitimate program, but when executed, it triggers an activity that may install a backdoor. Although some are secretly installed, other backdoors are deliberate and widely known. These kinds of backdoors have "legitimate" uses such as providing the manufacturer with a way to restore user passwords.

Many systems that store information within the cloud fail to create accurate security measures. If many systems are connected within the cloud, hackers can gain access to all other platforms through the most vulnerable system. Default passwords (or other default credentials) can function as backdoors if they are not changed by the user. Some debugging features can also act as backdoors if they are not removed in the release version. In 1993, the United States government attempted to deploy an encryption system, the Clipper chip, with an explicit backdoor for law enforcement and national security access. The chip was unsuccessful.

B AI Usage

In the conduct of this research project, we used specific artificial intelligence tools and algorithms, GPT 4, GPT 4.5, and Gemini 2.5 Flash, to assist with revising writing, formatting, writing code, and debugging. While these tools have augmented our capabilities and contributed to our findings, it's pertinent to note that they have inherent limitations. We have made every effort to use AI in a transparent and responsible manner. Any conclusions drawn are a result of combined human and machine insights. This is an automatic report generated with AI Usage Cards https://ai-cards.org (Wahle et al., 2023).

C Artifact Coverage

The datasets are only generated in English, their domain is drawn from the Helpful Instructions dataset, mostly everyday life questions.

D Licensing

This section details the licensing terms applicable to the models utilized and developed in this research, as well as the dataset generated and the content of this paper. Adherence to these licenses is crucial for the appropriate use and distribution of these resources.

Original Models

Our work builds upon two publicly available large language models:

- **Mistral 8B:** This model is subject to the **Mistral Research License**. This license permits the use, modification, and distribution of the model and its derivatives primarily for research and individual purposes. Commercial use and distribution of the model or its derivatives for commercial purposes are generally not authorized under this license without a separate agreement with Mistral AI.
- **Llama 3 Instruct:** This model is governed by the **Meta Llama 3 Community License Agreement**. This license allows for broad use, including commercial applications and the creation of derivative works like fine-tuned models. Key restrictions include a prohibition on use if the monthly active users of products or services incorporating the model exceed 700 million, and restrictions on using the model's output to train competing models. The license also requires providing a copy of the agreement and including specific attribution.

Fine-Tuned Models

The fine-tuned model weights developed as part of this research are released under licenses compatible with their respective base models. Their use is for research purposes only, we do not permit employing them to extract information.

Generated Dataset

The dataset generated through the course of this research is made publicly available under the **Creative Commons Attribution 4.0 International Public License (CC BY 4.0)**. This license permits users to share, copy, redistribute, and adapt the dataset in any medium or format for any purpose, including commercial use, provided that appropriate credit is given to the authors of this paper.