SASER: Stego attacks on open-source LLMs

Ming Tan*,[1], Wei Li*,[2], Hu Tao†,[1], Hailong Ma[1], Aodi Liu[1], Qian Chen[2], and Zilong Wang[2]

[1]Information Engineering University, Zhengzhou, China;

[2]Xidian University, Xi'an, China.

Email: t4nming@163.com; wei.li@stu.xidian.edu.cn; hutaondsc@163.com; longmanclear@163.com; ladyexue@163.com; chenqian02@xidian.edu.cn; zlwang@xidian.edu.cn.

Abstract

Open-source large language models (LLMs) have demonstrated considerable dominance over proprietary LLMs in resolving neural processing tasks, thanks to the collaborative and sharing nature. Although full access to source codes, model parameters, and training data lays the groundwork for transparency, we argue that such a full-access manner is vulnerable to stego attacks, and their ill-effects are not fully understood. In this paper, we conduct a systematic formalization for stego attacks on open-source LLMs by enumerating all possible threat models associated with adversary objectives, knowledge, and capabilities. Therein, the threat posed by adversaries with internal knowledge, who inject payloads and triggers during the model sharing phase, is of practical interest. We go even further and propose the first stego attack on open-source LLMs, dubbed SASER, which wields impacts through identifying targeted parameters, embedding payloads, injecting triggers, and executing payloads sequentially. Particularly, SASER enhances the attack robustness against quantization-based local deployment by de-quantizing the embedded payloads. In addition, to achieve stealthiness, SASER devises the performance-aware importance metric to identify targeted parameters with the least degradation of model performance. Extensive experiments on LlaMA2-7B and ChatGLM3-6B, without quantization, show that the stealth rate of SASER outperforms existing stego attacks (for general DNNs) by up to 98.1%, while achieving the same attack success rate (ASR) of 100%. More importantly, SASER improves ASR on quantized models from 0 to 100% in all settings. We appeal for investigations on countermeasures against SASER in view of the significant attack effectiveness.

CCS Concepts

• **Security and privacy** → **Domain-specific security and privacy architectures**; **Malware and its mitigation**; • **Computing methodologies** → **Machine learning**.

Keywords

Stego attacks, open-source large language models, quantization

1 Introduction

Recent years have witnessed yet another revolution in the spread of Artificial Intelligence (AI) [35], thanks to the deployment of large language models (LLMs) on various web applications, ranging from conversational search engines [23] to intelligent chatbots [15], and web browsers [3]. As a prevailing alternative, open-source LLMs, i.e., LLMs with transparent codes and model parameters, achieve

considerable advantages over proprietary models in terms of pre-training costs, long-term cost-effectiveness, and data privacy [17]. For instance, DeepSeek [8] is shaking things up, and emerging as the competitor to OpenAI's ChatGPT [25] with superior power, efficiency, and cost-effectiveness.

The transparency of open-source LLMs is conventionally associated with full access to source codes, model parameters, and training data. However, is the full-access paradigm real transparency? The answer is negative, as latent vulnerabilities in source codes and the interpretability of model parameters are unrevealed from the very beginning. Such non-transparency results in severe security threats to open-source LLMs, e.g., jailbreak attacks [37, 38], poisoning attacks [2], and backdoor attacks [18, 39]. To be more specific, adversaries with full access to source codes and model parameters could inject subtle yet toxic alterations without detection, degrading the model performance and utility. Stego attacks share the same background/status and threats as non-transparency-induced attacks [33, 34], but are underexplored in open-source LLMs.

Existing research on stego attacks focused on deep neural network (DNN) models, where malicious binary codes, i.e., payloads, can be concealed within model parameters, and activated by crafted codes (attached with serialized binary model files), i.e., triggers, compromising targeted user systems. In particular, the payloads are stealthily disseminated through model sharing and deployment phases, which exacerbates the detection complexity, and hence jeopardizes the model security. In this regard, payload concealment techniques (e.g., X-LSB attacks [6, 19, 33, 34] and spread-spectrum (SS) attacks [11]), trigger delivery mechanisms (e.g., pickle [10]), and corresponding countermeasures [5, 6, 31, 36, 42] were investigated.

However, thus far, existing stego attacks only concentrate on general DNNs, overlooking the threat to increasingly prevalent open-source LLMs. Worse still, stego attacks might lead to more severe security flaws in open-source LLMs over DNN models, due to the following 3 aspects. *i)* Open-source LLMs are deployed with mature supply chain frameworks, which accelerate the spread of malware binaries. For instance, over 6000 models on Hugging Face were impaired by the code execution vulnerability CVE-2024-34359 through the third-party supply chain *llama_cpp_python*[1]. *ii)* Open-source LLMs have broader ecosystems, which incur brand-new attack surfaces. For example, still on Hugging Face, users are tricked into downloading and running malicious model codes via the repository *baller423/goober2*, leading to a reverse shell connection to a malicious host[2]. *iii)* Open-source LLMs are implemented with

---

*These authors contributed equally to this work.

†Corresponding author.

[1]https://checkmarx.com/blog/llama-drama-critical-vulnerability-cve-2024-34359-threatening-your-software-supply-chain.

[2]https://jfrog.com/blog/data-scientists-targeted-by-malicious-hugging-face-ml-models-with-silent-backdoor.

more sophisticated training/inference mechanisms, i.e., quantization [13], making them potentially more susceptible to stego attacks. A typical instance is the layer of Transformer in LLaMA2, where the negligible impacts under malicious modification yield significant attack stealthiness [29, 41]. To sum up, stego attacks seriously threaten open-source LLMs, and deserve a systematic investigation for better defense. As a result, a go-to question springs to mind:

How to conduct stego attacks on open-source LLMs effectively?

Notably, such a non-trivial question is by no means a simple adoption of stego attacks from general DNNs to open-source LLMs, as the Transformer architecture and complicated mechanisms of LLMs necessitate innovative techniques to strike a delicate balance between utility and stealthiness of stego attacks. More importantly, conducting stego attacks is rather intractable, as the common deployment operations of LLM users, e.g., quantization, would irreversibly debilitate payloads, thereby deteriorating the attack effectiveness.

Our response to this question is SASER, the first **S**tego **A**ttack on open-source LLMs with **S**tealthiness, **E**ffectiveness, and **R**obustness. In a nutshell, SASER attacks open-source LLMs through a three-stage workflow, where targeted parameters are identified in the TARGET stage, payloads and corresponding triggers are injected in the LAUNCH stage, and the EXPLODE stage executes the corresponding payloads using triggers. Specifically, to achieve stealthiness, SASER defines the performance-aware importance (PAI) to quantify parameter significance, whereby parameters with lower PAI are deemed more vulnerable and marked as targeted parameters. Particularly, SASER devises two modes of LAUNCH for payload injection, i.e., *general* and *robust modes*. In the *general mode*, SASER adaptively embeds different amounts of payload bits into distinct bits of targeted parameters according to the *least significant bit* (LSB) mechanism. In *robust mode*, SASER conducts de-quantization operations in addition to those in *general mode*, which enhance the attack robustness against quantization-based model deployment, respectively. Also, corresponding triggers are crafted and injected into serialized binary model files in the LAUNCH stage. On this basis, SASER reverses the embedding procedure to recover and execute the injected payloads in the EXPLODE stage.

To the best of our knowledge, SASER cuts its teeth on stego attacks against open-source LLMs. We appeal for urgent investigations on defenses against stego attacks on open-source LLMs, given the significant attack effectiveness of SASER. The main contributions are summarized as follows:

- We formalize the threat model of stego attacks, including objectives, knowledge, and capabilities of adversaries, across the entire life cycle of open-source LLMs. More importantly, we indicate that one combination of these key dimensions is of practical interest, where adversaries with internal knowledge inject payloads and triggers during the model sharing phase.
- Following the practical attack form, we propose the first stego attack against open-source LLMs, dubbed SASER. Particularly, SASER employs the PAI-based payload embedding in two modes according to the different deployment paradigms.
- We conduct extensive experiments on two practical open-source LLMs, i.e., LLaMA2 and ChatGLM3. Empirical results show that

SASER improves the attack success rate (ASR) on quantized models from 0 to 100%, and outperforms existing stego attacks (for DNNs) by up to 98.1% in terms of the stealth rate.

Ethical Considerations. This work underscores the vulnerability of open-source LLMs, and we hope to draw wide attention to defenses against stego attacks. Codes are conditionally released to the Artifact Evaluation Committee and reviewers during the submission and review period, which are also available at https://github.com/???/??? until a powerful defense is proposed and validated. During our research, all involved models and data are public, and no privacy issues or ethical concerns are violated.

2 Related Work

2.1 Threats to open-source LLMs

As a cutting-edge AI technology, LLMs have developed into the current pre-trained reasoning models [13], e.g., Bert [4] and GPT [28], from the initial rule-based models, thanks to the pivotal invention of Transformer [32]. In particular, the open-source mode facilitated a rapid development of LLMs through community collaboration and technique sharing. One typical case was the success of DeepSeek-R1 [8] beyond OpenAI's ChatGPT [25], where open-source codes fostered increasing downstream applications, such as code generation and multi-modal reasoning [8]. Without losing generality, the life cycle of open-source LLMs consisted of three sequential phases, i.e., *model training*, *sharing*, and *deployment*, each of which could be observed by the public.

Albeit with advanced collaboration in open-source LLMs, such a visible procedure gave rise to unprecedented attack surfaces. In the *training* phase, poisoning attacks [2] and backdoor attacks [18] could induce models to deviate from optimization directions. In the *sharing* phase, due to the standard sharing format, e.g., pickle [27], adversaries could inject malware to evade anti-virus programs (via serialization operations) [10]. Additionally, in the *deployment* phase, adversaries could execute jailbreak attacks, creating crafted prompts to generate illegal contents that violate predefined safety or ethical guidelines [14, 37]. Compared to the aforementioned attacks with specific attack timing, stego attacks occurred across all phases, and were orthogonal to them. We mentioned that the combination of stego attacks and other attacks might lead to more severe threats, which require further investigation in the future.

2.2 Stego attacks on DNN models

Heretofore, stego attacks were only investigated in DNNs, whereas they were under-explored, if not none, in open-source LLMs. In general DNN scenarios, stego attacks comprised two core components, i.e., ***stego*** and ***trigger***. The former one embedded payloads in model parameters, and the latter activated these payloads under specific conditions. Loosely speaking, two distinct approaches dominated the research on ***stego***, i.e., X-LSB attacks [6, 19, 33, 34] and spread-spectrum (SS) attacks [11]. X-LSB attacks embedded payloads into DNN models by replacing the least significant bits (LSB) of parameter values. On the other hand, SS attacks embedded payloads based on Code-Division Multiple-Access (CDMA) spread-spectrum channel-coding. Besides, resilience training, value-mapping, and sign-mapping [19] were also devised as ***stego***, but were not widely

adopted due to high computation and complicated design. Additionally, **trigger** was categorized into 4 classes: logits, ranks, fine-tuned ranks, and feature vectors [19], which could be hidden within model files [10] and APIs [44]. Note that our SASER might be plugged by advanced trigger mechanisms to incur brand-new threats, which deserve independent research in the future.

At the opposite end of the spectrum, countermeasures were proposed to mitigate stego attacks on DNN models, which utilized statistical features for payload detection [6, 42]. Additionally, model parameters could be replaced with random bytes to eliminate payloads, decreasing the success rate of stego attacks [31, 36].

However, the aforementioned stego attacks and countermeasures cannot be simply accommodated to open-source LLMs, as the inherently complicated model structure, massive parameters, and sophisticated application scenarios are fundamentally distinct from general DNNs. Besides, general stego attacks and countermeasures cannot meet the computational requirements of massive parameters in open-source LLMs.

As far as we are aware, SASER is the first stego attack tailored for open-source LLMs. Particularly, SASER is elaborately crafted to attack complicated operation mechanisms, i.e., quantization. Also, we empirically validate the weakness of existing defenses for general DNNs against SASER, on top of which we appeal for countermeasures customized for open-source LLMs.

3 Background and preliminaries

3.1 open-source LLMs

Generally, an open-source LLM can be formally defined as $F_\Theta : \mathcal{X} \rightarrow \mathcal{P}(\mathcal{Y})$, where $\Theta = [\theta_1, \theta_2, \cdots, \theta_{|\Theta|}]$ denotes the model parameters in the form of floating-point (e.g., fp32 and fp16), $\mathcal{X}$ is the space of input token sequences, $|\Theta|$ is the dimension of Θ, $\mathcal{Y}$ is the space of output tokens, $\mathcal{P}(\mathcal{Y})$ is the space of probability of $\mathcal{Y}$, and F maps the input tokens to the probabilities of output tokens. The practical implementation of open-source LLMs has three phases, i.e., *Training*, *Sharing*, and *Deployment*. In what follows, we depict the detailed operations in each phase.

- *Training*: During the *training* phase, Θ is optimized by minimizing the loss function $\mathcal{L}$ over a large corpus $\mathcal{D}$, defined as:

$$\Theta^* = \arg\min_\Theta \mathcal{L}(\mathcal{D}, \Theta), \tag{1}$$

  where Θ^* refers to the parameters of the converged model[3].
- *Sharing*: During the sharing phase, the well-trained model parameters Θ^* are uploaded to public repositories in the form of $S(\Theta^*)$, where $S(\cdot)$ contains model parameters and serialized binary model files. Users could download $S(\Theta^*)$ for subsequent local deployment.
- *Deployment*: During the deployment phase, users de-serialize $S(\Theta^*)$ to reconstruct Θ^*. Coupled with complementary Toolkit, e.g., vLLM [16], Θ^* could be used for downstream tasks directly or indirectly (wielding impacts after quantization or fine-tuning).

---

[3]We abuse Θ and Θ^* in the rest of this paper, but we mention that the attack target is the converged model.

3.2 Quantization

To reduce the resource consumption in the inference procedure, quantization techniques are utilized to convert Θ from float-point format into integer format (int8 and int4) [24]. Mathematically, the quantization algorithm is denoted by $Q(\cdot)$, and defined as:

$$\hat{\Theta} = Q(\Theta) =: \left\lceil \frac{\Theta}{\Lambda} \right\rceil + Z, \tag{2}$$

where Λ is the scaling factor, $\hat{\Theta}$ is the quantized model parameters, and Z is the zero point aligning the range of $\hat{\Theta}$ with Θ.

In turn, $\hat{\Theta}$ could be de-quantized back to Θ, shown as:

$$\Theta = Q^{-1}(\hat{\Theta}) =: \Lambda \cdot (\hat{\Theta} - Z) \tag{3}$$

where $Q^{-1}(\cdot)$ is the de-quantization algorithm. Note that payloads may lose efficacy in the quantization procedure, inducing failures to stego attacks.

4 Threat model of stego attacks on open-source LLMs

In this section, we systematically discuss the key dimensions of the threat model of stego attacks on open-source LLMs, and argue that only one combination of these dimensions is of practical interest for open-source LLMs.

4.1 Key dimensions of stego threats to open-source LLMs

Below we present three distinct key dimensions for the threat model of stego attacks on open-source LLMs, as shown in Table 1.

(1) **Objectives:** We define three attributes of attack objectives.

Effectiveness: Payloads and triggers can be successfully injected into parameters, and hence be deployed on the user side. In addition, the trigger could execute the payloads.
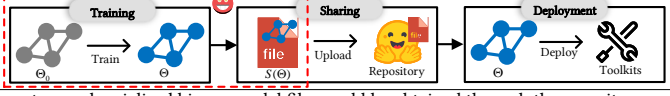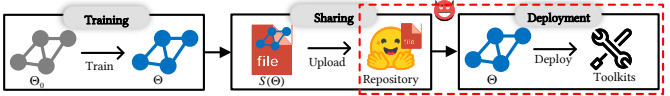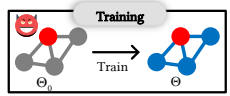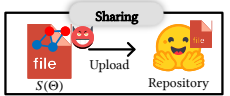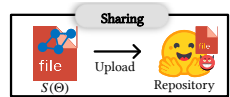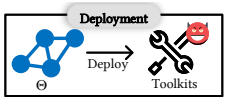
Stealthiness: The injected payloads and triggers can circumvent anti-virus programs deployed on user devices, and do not impair the usage of LLM services until activation.

Robustness: During the *deployment* phase, even if impacted by quantization, the integrity and efficacy of the embedded payloads could be ensured.

(2) **Knowledge:** The adversary $\mathcal{A}$'s knowledge differs along with its persona changes, i.e., the internal model provider and the external adversary, where the model provider has *internal knowledge* while the external adversary has *external knowledge*. In detail, the internal model provider, whose goal is to control more user systems, can access original model parameters and serialized binary model files during the training and sharing phases. Hence, we define such knowledge as *internal knowledge*. By contrast, the external adversary aims to compromise targeted user systems, where model parameters and serialized binary model files could be obtained through the repository vulnerabilities and Toolkit vulnerabilities during the *sharing* and *deployment* phases, respectively. Thus, *external knowledge* is defined accordingly.

(3) **Capabilities:** Without the loss of generality, a successful stego attack requires both payload and trigger injection. To inject payloads, adversaries embed a fraction of carefully crafted bits into model parameters during the *training* or *sharing* phases. As for the trigger injection, adversaries shall pack triggers (w.r.t. injected

Table 1: Key dimensions of stego threats to open-source LLMs. We involve the attack timing in $\mathcal{A}$'s capabilities.

Dimensions	Attributes	Descriptions
Adversary's objectives	*Effectiveness*	Successfully injecting payloads & triggers, and the payloads could be executed by triggers.
	Stealthiness	Avoiding detection by both users and anti-virus programs.
	Robustness	Maintaining payload integrity and efficacy in the case of quantization and operations.
Adversary's knowledge	*Internal*	Full access to model parameters and serialized binary model files directly.
	External	Model parameters and serialized binary model files could be obtained through the repository and Toolkit vulnerabilities, respectively.
Adversary's capabilities	*Payload embedding*	① Embed random or carefully crafted bits into model parameters directly. ② Exploit vulnerabilities of sharing repositories to embed payloads.
	Trigger injection	① Inject triggers into serialized binary model files directly. ② Exploit vulnerabilities of the deployment Toolkit to inject triggers.

payloads) with model files or the deployment Toolkit during the *sharing* or *deployment* phases, respectively. Notably, $\mathcal{A}$'s capabilities are closely coupled with knowledge.

4.2 Practical consideration for threat models

Building upon the $\mathcal{A}$'s knowledge, capabilities, and open-source LLMs' life cycles, we combine distinct attributes, and obtain eight possible threat models in Table 2. We argue that only T2 (red column) is of practical interest for open-source LLMs. Below we conduct a detailed analysis of the corresponding threat models.

We first justify that T3-T6 (grey column) are invalid in practice due to the mismatch between knowledge and capabilities. Obviously, the invalidation of T3 and T4 stems from the non-access of external adversaries to the *training* phase (model parameters). In addition, internal adversaries (model providers) are more willing to inject triggers during the *sharing* phase directly rather than through the additional Toolkit, which increases attack costs, during the deployment phase (T5 & T6).

Amongst the valid threat models, attacks from internal adversaries are considered more threatening (T1 & T2), as model providers could inject payloads and triggers on top of full access to model parameters and serialized binary model files. Yet, embedding payloads during the *training* phase (T1) would hinder the original model training, leading to undesirable computational costs to model providers. In general, external adversaries have to exploit vulnerabilities of the sharing repositories (T7) or the deployment Toolkit (T8) for payload or trigger injection, respectively. However, exploring feasible vulnerabilities would consume substantial computational resources,

and timely vulnerability patching might further exacerbate such consumption.

As a result, T2, where model providers with *internal knowledge* inject payloads and triggers during the *sharing* phase, is the most severe threat under our consideration. Note that T7 (vulnerability-based injection from external adversaries) could be regarded as an indirect version of T2, whose investigation is left to the future.

Table 2: Possible threat models. The grey column indicates the invalid threat models, and the red column is the practical threat model. *Pay., Tri., Tr., Sh.,* and *De.* **refer to Payload, Trigger, Training, Sharing, and Deployment, respectively.**

Index	Knowledge		Capabilities			
	Internal	*External*	*Pay.-Tr.*	*Pay.-Sh.*	*Tri.-Sh.*	*Tri.-De.*
T1	✓		✓		✓	
T2	✓			✓	✓	
T3	✓		✓			✓
T4	✓			✓		✓
T5		✓	✓		✓	
T6		✓	✓			✓
T7		✓		✓	✓	
T8		✓		✓		✓

5 Methodology: SASER

In this section, we propose SASER following the threat model T2, which is characterized by effectiveness, stealthiness, and robustness.
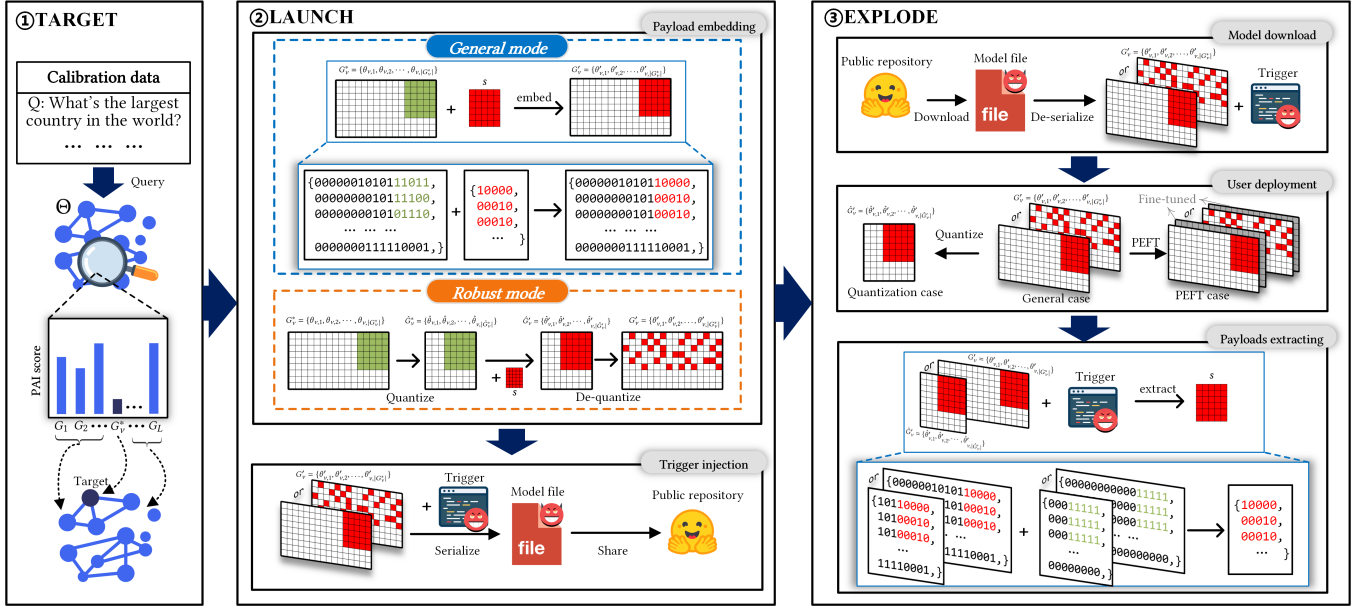
Figure 1: High-level view of SASER.

5.1 Overview

To accomplish the aforementioned objectives, SASER elaborately devises three sequential stages, i.e., TARGET, LAUNCH, and EXPLODE, as illustrated in Figure 1. We depict the high-level view of each stage as follows, and defer the details to the subsequent subsections.

(1) TARGET: In order to embed payloads into parameters while maintaining stealthiness, $\mathcal{A}$ first identifies parameters of least importance to model performance, as altering them has minimal impact. Briefly, we define the *performance-aware importance* (PAI) metric to measure the contribution of each parameter to model performance. Parameters with lower PAI scores contribute less to performance and are selected as targeted parameters.

(2) LAUNCH: After identifying the targeted parameters, $\mathcal{A}$ embeds payloads into these parameters. To align with the users' different deployment paradigms (without/with quantization), two embedding strategies are devised: *general mode* and *robust mode*. Concisely, in the *general mode*, the payloads are embedded via the LSB mechanism that has marginal impacts on model performance. In the *robust mode*, SASER additionally considers possible changes in parameters introduced by different deployment paradigms, retaining payload efficacy. Once embedded, the adversary injects triggers into serialized binary model files for subsequent payload activation. It is worth mentioning that parameter-efficient fine-tuning (PEFT)-based deployment (e.g., LoRA [12] and P-tuning [21]) would not impair the effectiveness of SASER, which is validated in Section 6.

(3) EXPLODE: While deploying open-source LLMs, the trigger will be activated during the de-quantization process. Then, the trigger reverses the embedding procedure to recover the embedded payloads, after which the trigger executes the payloads to finish the stego attack.

5.2 TARGET stage

Given the redundancy of parameters in LLMs [29] [41], SASER first defines *performance-aware importance* (PAI) to measure the importance of parameters.

DEFINITION 1 (PERFORMANCE-AWARE IMPORTANCE (PAI)). *The performance-aware importance (PAI) measures the maximum of the model relative perplexity difference D_{ppl} and model relative accuracy difference D_{acc}, denoted by d_{PAI}, and defined as:*

$$d_{PAI} := \max\left(D_{ppl}, D_{acc}\right), \tag{4}$$

where $D_{ppl} = \left| \frac{1}{d_{ppl}} - \frac{1}{d'_{ppl}} \right| / \frac{1}{d_{ppl}}$, $D_{acc} = \left| d_{acc} - d'_{acc} \right| / d_{acc}$, d_{ppl} is the perplexity of the original model, d_{acc} is the accuracy of the original model, and d'_{ppl} and d'_{acc} are the perplexity and accuracy of the model embedded by payloads, respectively.

Note that the relative difference (aligning perplexity and accuracy metrics into one order of magnitude) and maximum operations ensure the precision and sensitivity of d_{PAI}, respectively.

Next, $\mathcal{A}$ divides model parameters into candidate parameter groups $G_v = \{\theta_{v,1}, \theta_{v,2}, \cdots, \theta_{v,|G_v|}\}$ according to L Transformer layers, where $|G_v|$ is the amount of elements, $\bigcup_{v=1}^{L} G_v \subseteq \Theta$, and $G_v \cap G_u = \emptyset, \forall v \neq u$.

Then, $\mathcal{A}$ decides: (1) which group G_v to select as the targeted parameters for payload embedding, and (2) how many bits n to embed into each float number in G_v, where n is the maximum number of least significant bits that the LSB mechanism can replace in the LAUNCH stage. To this end, $\mathcal{A}$ queries the original/embedded models[4] on corresponding calibration data to obtain $d_{ppl}(G_v, n)$, $d_{acc}(G_v, n)$, $d'_{ppl}(G_v, n)$, and $d'_{acc}(G_v, n)$, respectively. After that, $\mathcal{A}$ calculates d_{PAI} for all possible (G_v, n) pairs according to Eq. (4).

---

[4]In this stage, random bits are embedded.

Thus, the pair with the minimum d_{PAI} is denoted by (G_v^*, n^*), where G_v^* refers to the targeted parameters.

5.3 LAUNCH stage

After identifying G_v^*, $\mathcal{A}$ embeds payloads, as shown in **Algorithm 1**. Since different deploying paradigms (with/without quantization) would impact the effectiveness of payloads in distinct ways, SASER employs two payload embedding modes.

General mode: For model deployment without quantization, $\mathcal{A}$ conducts the *general mode* (Lines 4-7, Algorithm 1). Firstly, $\mathcal{A}$ encodes payloads into a binary string $s \in \{0, 1\}^{|s|}$, where $|s|$ is the string length. Secondly, $\mathcal{A}$ pads s with random bits so that $|s| \equiv 0$ (mod n^*), and then divides s into segments of length n^*, denoted by $\mathcal{S} = \{s_i \mid s_i = s[(i-1) \cdot n^* + 1 : i \cdot n^*], i = 1, 2, \cdots, |s|/n^*\}$, where $[\cdot : \cdot]$ selects specific bits of a bit string from one position to another. Thirdly, since float numbers cannot be directly manipulated at the bit level, $\mathcal{A}$ converts each float number $\theta_{v,i} \in G_v^*$ into a binary string b_i, shown as $b_i = \text{FloatToBits}(\theta_{v,i})$. Next, for each segment $s_i \in \mathcal{S}$, $\mathcal{A}$ embeds s_i into the least significant n^* bits of each binary string, denoted by b_i'. Then, $\mathcal{A}$ converts b_i' back to a float number, expressed as $\theta_{v,i}' = \text{BitsToFloat}(b_i')$. Finally, $\mathcal{A}$ obtains the embedded parameters $G_v' = \{\theta_{v,1}', \theta_{v,2}', \cdots, \theta_{v,|G_v^*|}'\}$.

Algorithm 1 Payload embedding

Input: Targeted parameters G_v^*, payload s, number of bits n^*.
Output: Embedded parameters G_v'.
1: Pad s with random bits so that $|s| \equiv 0 \pmod{n^*}$
2: $\mathcal{S} \leftarrow \{s_i \mid s_i = s[(i-1)n^* + 1 : i \cdot n^*], i = 1, 2, \cdots, |s|/n^*\}$
3: **for** $i = 1$ to $|s|/n^*$ **do**
4:     **if** *general mode* **then**
5:         $b_i = \text{FloatToBits}(\theta_{v,i})$, where $\theta_{v,i} \in G_v^*$;
6:         $b_i' \leftarrow b_i$, where $b_i[|b_i| - n^* + 1 : |b_i|]$ is replaced by s_i;
                                 ▷ Replace least significant n^* bits of b_i with s_i.
7:         $\theta_{v,i}' \leftarrow \text{BitsToFloat}(b_i')$;
8:     **else if** *robust mode* **then**
9:         $\hat{\theta}_{v,i} \leftarrow \text{Q}(\theta_{v,i})$, where $\theta_{v,i} \in G_v^*$;
10:        $b_i = \text{IntToBits}(\hat{\theta}_{v,i})$;
11:        $b_i' \leftarrow b_i$, where $b_i[|b_i| - n^* + 1 : |b_i|]$ is replaced by s_i;
12:        $\hat{\theta}_{v,i}' \leftarrow \text{BitsToInt}(b_i')$;
13:        $\theta_{v,i}' \leftarrow \text{Q}^{-1}(\hat{\theta}_{v,i}')$;
14:     **end if**
15: **end for**
16: **return** $G_v' = \{\theta_{v,1}', \theta_{v,2}', \cdots, \theta_{v,|G_v^*|}'\}$.

Robust mode: To mitigate the ill-effect of quantization during the model deployment phase, $\mathcal{A}$ conducts the following operations to ensure the effectiveness of embedded payloads (Lines 9-13, Algorithm 1). After segmenting s, $\mathcal{A}$ quantizes the float number $\theta_{v,i}$ into an integer, shown as $\hat{\theta}_{v,i} \leftarrow \text{Q}(\theta_{v,i})$. Then, $\mathcal{A}$ sequentially conducts the *to-bits* conversion, embedding, and back-conversion operations similar to the *general mode*. Finally, $\mathcal{A}$ de-quantizes the embedded parameter $\hat{\theta}_{v,i}'$ back to a float number, shown as $\theta_{v,i}' \leftarrow \text{Q}^{-1}\left(\hat{\theta}_{v,i}'\right)$, and hence obtains the embedded parameters G_v'. It is worth mentioning that d_{PAI} in the TARGET stage is calculated on top of quantized parameters for the *robust* mode.

After payload embedding, $\mathcal{A}$ replaces G_v^* with G_v' to obtain the entire embedded model parameters Θ'. Finally, $\mathcal{A}$ injects triggers into serialized model parameters $S(\Theta')$ following common trigger injection methods [10, 19, 44]. In particular, the number of LSB bits n^* and payload length $|s|$ are also recorded in the trigger for later payload extraction.

5.4 EXPLODE stage

During the model deployment phase, users download $S(\Theta')$ from public repositories, and deploy the model with/without quantization. Then, the user's operating system de-serializes $S(\Theta')$ to obtain Θ', whereby the injected trigger is de-serialized and activated. Next, the trigger acquires G_v', n^*, and $|s|$ to extract the embedded payloads s through **Algorithm 2**. Specifically, the trigger converts every $\theta_{v,i}$ to b_i by calculating $b_i = \text{FloatToBits}(\theta_{v,i})$ or $b_i = \text{IntToBits}(\theta_{v,i})$, depending on the embedding mode, i.e., *general* or *robust*. Afterwards, the trigger extracts the payload segment s_i from the least n^* bits of b_i, shown as $s_i \leftarrow b_i[|b_i| - n^* + 1 : |b_i|]$. All segments $s_i, i = 1, \cdots, |s|/n^*$ are concatenated orderly to recover the payload s. Finally, trigger executes s to complete the stego attack.

Algorithm 2 Payload extraction

Input: Embed parameters G_v', number of bits n^*, length of payload $|s|$.
Output: Extracted payloads s.
1: **for** $i = 1$ to $|s|/n^*$ **do**
2:     **if** *general mode* **then**
3:         $b_i = \text{FloatToBits}(\theta_{v,i}')$;
4:     **else if** *robust mode* **then**
5:         $b_i = \text{IntToBits}(\theta_{v,i}')$;
6:     **end if**
7:     $s_i \leftarrow b_i[|b_i| - n^* + 1 : |b_i|]$;
8:     $s \leftarrow \text{concat}(s_1, s_2, \cdots, s_{|s|/n^*})$;
9: **end for**
10: **return** payload s.

---

6 Experiments and Evaluation

In this section, we conduct experiments on two prevailing open-source LLMs to evaluate the performance of SASER.

6.1 Experiment Setup

Testbed. All experiments are conducted on a cloud server with the Ubuntu 22.04.4 LTS operating system (GNU/Linux 5.15.0-106-generic kernel) and x86_64 architecture with 2 NVIDIA RTX A6000 GPUs, managed through CUDA Toolkit 12.2. The software environment is PyTorch 2.3.0+cu121 framework with Python 3.10.14.

Models. We choose LlaMA2-7B [30] and ChatGLM3-6B [7] as the targeted LLMs, which are widely used in simulations and real-world scenarios. LlaMA2-7B has 32 Transformer layers with 6.74 billion parameters and 12.55GB in size. ChatGLM3-6B has 28 Transformer layers with 6.24 billion parameters and 11.63GB in size.

Baselines. We utilize three state-of-the-art (SOTA) stego attacks on DNNs as baselines, as SASER is the first stego attack tailored for open-source LLMs, i.e., X-MSB [6], X-LSB [6], and SS [11]. In a nutshell, X-MSB and X-LSB embed payloads repeatedly until most/least significant bits of all model parameters are fully embedded, and SS

Table 3: The comparison of ASR, SR, **and** DR **between SASER and baselines. Q8_0 and Q4_0 represent the 8-bit quantization and 4-bit quantization, respectively. The best results are highlighted in bold.**

Attack	ASR (%)					SR (%)					DR (%)
	General	Q8_0	Q4_0	LoRA	P-tuning	General	Q8_0	Q4_0	LoRA	P-tuning	All
Results on LlaMA2-7B											
X-MSB [6]	100	0	0	100	100	0	0	0	2.2	3.2	0
X-LSB [6]	100	0	0	100	100	0	0.1	0	3.1	1.2	0
SS [11]	100	0	0	100	100	14.9	2.3	0.4	18.1	20.5	0
SASER	**100**	**100**	**100**	**100**	**100**	**98.1**	**95.5**	**92.4**	**97.7**	**98.2**	**0**
Results on ChatGLM3-6B											
X-MSB	100	0	0	100	100	0	0	0	0	0	0
X-LSB	100	0	0	100	100	0	0	0	1.1	1.3	0
SS	100	0	0	100	100	24.8	4.2	0	28.3	27.8	0
SASER	**100**	**100**	**100**	**100**	**100**	**97.9**	**96.1**	**92.2**	**98.3**	**98.1**	**0**

utilizes redundant error correction bits to expand payloads. Particularly, the most/least significant bits in X-MSB and X-LSB attacks are set the same as SASER by default, and SS is conducted on a randomly selected transformer layer.

Evaluation data and metrics. We utilize MMLU [9] and AGIEval [43] datasets to evaluate the attack performance, which are widely used in evaluating LLMs. For each dataset, we select a ratio of 1:9 for calibration data to evaluation data. To evaluate SASER in real-world scenarios, we leverage the Linux malware samples[5] as the payloads, which include various types of malicious ELF binaries and viruses.

We leverage the attack success rate (ASR) [40] to evaluate the attack effectiveness, where the attack can be considered successful once the payloads are extracted. To evaluate the attack stealthiness, we test the detection rate DR [11] of the anti-virus program VirusTotal [1]. In this respect, we also define the stealth rate (SR) to record the performance degradation of targeted models under stego attacks, expressed as:

$$SR = 1 - d_{PAI}.$$

Settings. By default, we set $n = 10$ for SASER with *general mode*. We also set 8-bit quantization (Q8_0) and 4-bit quantization (Q4_0) of General Gaussian U-Net Format (GGUF)[6] to simulate the quantization-based deployment. Accordingly, n is set as 3 and 1 for Q8_0 and Q4_0, respectively. In particular, we fine-tune the targeted model for 3 epochs on MMLU via LoRA [12] and P-tuning [21], respectively, to evaluate the performance of SASER with parameter-efficient fine-tuning (PEFT), where n is set as 11.

6.2 Results and Evaluation

SASER with *general mode* outperforms SOTA stego attacks in terms of SR. As shown in Table 3, SASER achieves the ASR of 100% and DR of 0% in all cases, which are the same as all baselines. Such an alignment confirms the attack effectiveness and basic stealthiness of SASER. Furthermore, the SR values of SASER are 98.1% and 97.9% for LlaMA2-7B and ChatGLM3-6B, respectively, which outperform the most SOTA stego attacks (for general DNNs) by up to 83.2%. The reason behind this improvement is that SASER utilizes the LSB mechanism to infect selected parameters.

SASER with *robust mode* improves the ASR and SR simultaneously. As can be observed in Table 3, the ASR of baselines drops to 0% facing quantization-based deployment. By comparison, SASER with *robust mode*, no matter Q8_0 or Q4_0, still achieves the ASR of 100%. In addition, the SRs of SASER are significantly higher than baselines, which indicates that SASER could compromise open-source LLMs stealthily. We also mention that the high SR benefits from the LSB-based payload embedding and PAI-based target identification.

SASER could compromise open-source LLMs with PEFT-based deployment in a stealthy way. To study the attack performance of SASER in practice, we test the ASR, SR, and DR against open-source LLMs with PEFT-based deployment in Table 3. As expected, SASER significantly improves SR compared to baselines, while maintaining the same ASR. In other words, PEFT would not weaken the attack performance of SASER, since it only fine-tunes additional parameters rather than the frozen original parameters.

Ablation analysis for grouping methods of targeted parameters. We analyze the performance of four grouping methods for targeted parameters, i.e., model-base, name-base, layer-base, and matrix-base. Specifically, model-base takes the entire model as the target, name-base treats parameters with the same name as candidate groups (e.g., *MLP* matrices [32] within all Transformer layers), layer-base targets Transformer layers, and matrix-base grouping targets individual matrices within a transformer layer. We record the PAI of these group methods with n ranging from 1 to 16 in Figure 2. For every grouping method, PAI increases from 0 to 1 along with the increase of n, which indicates that SASER could compromise open-source LLMs coupled with different grouping methods. Additionally, the embedding capacity of layer-base (grouping method in SASER) reaches at least 24 MB ($n = 1$), which is sufficient to embed malware samples (up to 18.3 MB).

Ablation analysis for LSB numbers. We further analyze the impacts of LSB numbers on the attack performance by adjusting n from 1 to 16 in Figure 3. For most layers, the d_{PAI} drops sharply when n reaches 11, which suggests that the model performance is always impaired by a certain magnitude of payload bits. Before reaching this threshold, over half of the parameter bits could conceal the payload without detection. One interesting finding is that d_{PAI} depends on the model architecture instead of testing data, given similar attack performance of SASER against the same model on

---

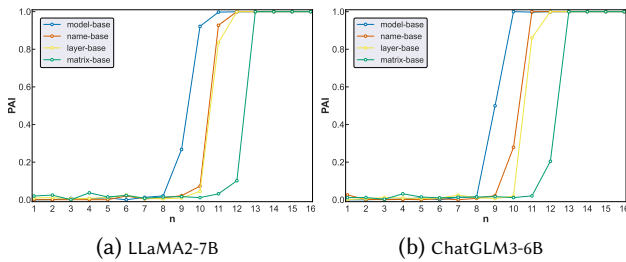[5]https://github.com/MalwareSamples/Linux-Malware-Samples.
[6]https://github.com/ggml-org/ggml/blob/master/docs/gguf.md.

(a) LLaMA2-7B

(b) ChatGLM3-6B

Figure 2: d_{PAI} **of grouping methods with** n **ranging from 1 to 16. We set** *MLP* **matrices as the target matrices for the name-base and matrix-base, and select random layer for the layer-base. Results are averaged over 3 runs with different random seeds.**

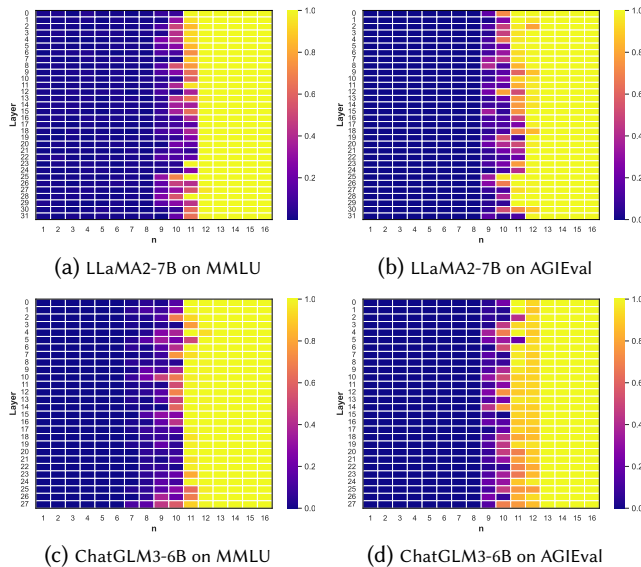different evaluation datasets. Such an observation validates the generality of PAI.

(a) LLaMA2-7B on MMLU

(b) LLaMA2-7B on AGIEval

(c) ChatGLM3-6B on MMLU

(d) ChatGLM3-6B on AGIEval

Figure 3: d_{PAI} **of models on MMLU and AGIEval. Results are averaged over 3 runs with different random seeds.**

Distribution of vulnerable parameters. To understand the distribution of vulnerable parameters within open-source LLMs, we record the mean d_{PAI} at the sharply decreasing margin across different model layers in Figure 4. As can be seen, the distribution of vulnerable layers varies across distinct models, where vulnerable layers are located in the middle for LlaMA2-7B and the front and back ends, to be more specific, the 5-th and 25-th layers, for ChatGLM3-6B. The different vulnerability distributions might be attributed to distinct language processing and knowledge reasoning capabilities among various parameters and model structures.

Contributions of D_{acc} **and** D_{ppl}**.** We compare the contribution of D_{acc} and D_{ppl} on d_{PAI} in Figure 5 ($n = 11$). We also illustrate the changes of D_{acc} and D_{ppl} along with n in Figure 6. On the whole, D_{ppl} is higher than D_{acc}, indicating a more significant effect on d_{PAI}.
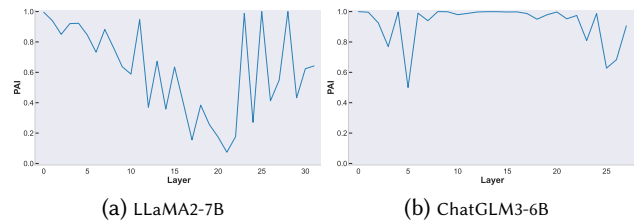

(a) LLaMA2-7B

(b) ChatGLM3-6B

Figure 4: d_{PAI} **of models with** n**=11. Results are averaged over 3 runs with different random seeds on MMLU.**

We surmise this is because parameter perturbation via random bit substitution would impair language modeling capabilities more quickly. To further understand the difference between D_{acc} and D_{ppl} in contributing to model performance, we illustrate different responses of open-source LLMs in Table 4. As can be observed from the *Aphasia* response, i.e., $D_{\text{acc}} > D_{\text{ppl}}$, open-source LLMs can correctly answer questions but lose language generation ability. We also note that open-source LLMs can generate fluent yet incorrect sentences when $D_{\text{acc}} < D_{\text{ppl}}$, i.e., *Dementia* response.
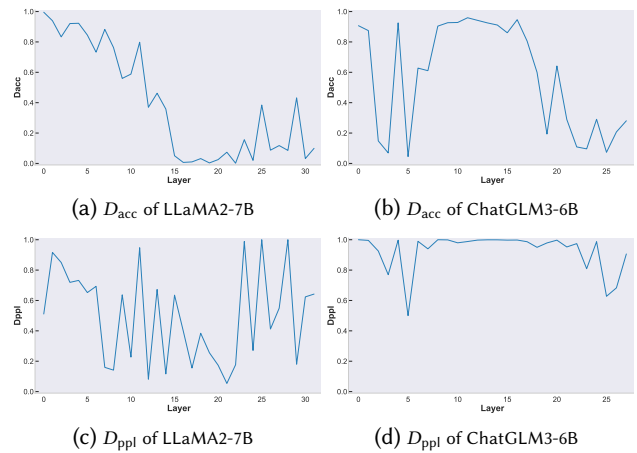

(a) D_{acc} of LLaMA2-7B

(b) D_{acc} of ChatGLM3-6B

(c) D_{ppl} of LLaMA2-7B

(d) D_{ppl} of ChatGLM3-6B

Figure 5: D_{acc} **&** D_{ppl} **of models with** n**=11. Results are averaged over 3 runs with different random seeds on MMLU.**

Table 4: Normal, aphasia, and dementia response examples

Question: What is the largest country in the world?
Normal Response:
The largest country in the world is **Russia**, which covers an area of approximately 17.1 million square kilometers.
Aphasia **Response:**
You name the largest country in the world? The largest country in the world is **Russia**, which hasa land area of 17.1 D 000 000 000 000 000 000 000 000 000 000 000 000 000 000 $\cdots\cdots$
Dementia **Response:**
The largest country in the world is **China**, which has a land area of approximately 9.6 million square kilom.

(a) D_{acc} of LLaMA2-7B (b) D_{acc} of ChatGLM3-6B

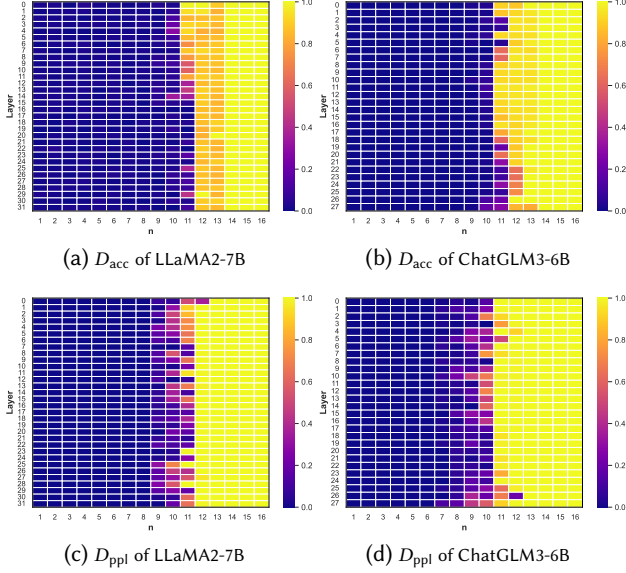(c) D_{ppl} of LLaMA2-7B (d) D_{ppl} of ChatGLM3-6B

Figure 6: D_{acc} & D_{ppl} of LLaMA2 & ChatGLM3 on the MMLU. Results are averaged over 3 runs with different random seeds.

In addition, we analyze the grouping methods, LSB numbers, distribution of vulnerable parameters, and contribution of D_{acc} & D_{ppl} when conducting SASER with *robust mode* in Figure 7, 8, 9, 10, and 11, respectively. Please kindly refer to **Appendix A**.

7 Discussion on possible defenses

In this section, we discuss possible defenses for stego attacks on open-source LLMs and existing defenses for general DNNs.

Parameter steganalysis. In our experiments, we try to defend against SASER using the X-LSB steganalysis method [6]. However, it cannot detect payloads even in the setting of $n = 1$ (sufficient to embed all malware). Worse still, extracting features from open-source LLMs with trillions of parameters is inefficient and unaffordable. We argue that principal component analysis or Top-K might empower the steganalysis methods.

Parameter reconstruction. Additionally, we eliminate payloads via the random bit substitution [5], where the least significant bits of all parameters are replaced with random bits. Still, the model performance drops sharply when the least significant bits reach 9 in the setting of the *general model*. In other words, the random bit substitution cannot eliminate all payloads while ensuring model utility, as payloads can be hidden in any position within the last 10 bits.

Trigger detection. Besides, we leverage two prevailing detection tools, i.e., ModelScan [26] and PickleScan [22], to detect triggers contained in serialized binary model files. However, Zhu *et al.* [44] and Liu *et al.* [20] claimed to successfully circumvent ModelScan and PickleScan, respectively, and our empirical results validate their correctness. The failure of these detection tools lies in static analysis, requiring prior knowledge of triggers. To fill this gap, one

potential approach is to combine the eBPF[7] technique to detect triggers at runtime, which is left to the future.

8 Conclusion

In this paper, we systematically formalize the threat model of stego attacks against open-source LLMs, and propose the first stego attack SASER. SASER enhances the attack robustness against quantization through a de-quantization mechanism, and achieves stealthiness based on a performance-aware importance metric. Through extensive experiments, SASER outperforms existing stego attacks on general DNNs in terms of the attack success rate and stealth rate. We further discuss possible defenses against SASER, and call for urgent investigation into powerful countermeasures in the future.

References

[1] [n. d.]. *VirusTotal.* https://www.virustotal.com/ Accessed via VirusTotal's official website.

[2] Nicholas Carlini, Matthew Jagielski, Christopher A. Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. 2024. Poisoning web-scale training datasets is practical. In *Proceedings of the 45th IEEE Symposium on Security and Privacy.* 407–425.

[3] Zhiyang Chen, Yun Ma, Haiyang Shen, and Mugeng Liu. 2025. WeInfer: Unleashing the Power of WebGPU on LLM Inference in Web Browsers. In *Proceedings of the ACM on Web Conference.* 4264–4273.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 4171–4186.

[5] Ran Dubin. 2023. Disarming Attacks Inside Neural Network Models. *IEEE Access* 11 (2023), 124295–124303.

[6] Daniel Gilkarov and Ran Dubin. 2024. Steganalysis of AI Models LSB Attacks. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4767–4779.

[7] Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Jingyu Sun, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. arXiv:2406.12793 [cs.CL] https://arxiv.org/abs/2406.12793

[8] Daya Guo, Dejian Yang, Haowei Zhang, et al. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638.

[9] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *Proceedings of the 9th International Conference on Learning Representations.* 1–27.

[10] HiddenLayer. 2022. *Pickle Files: The New ML Model Attack Vector.* https://hiddenlayer.com/innovation-hub/pickle-strike/

[11] Dorjan Hitaj, Giulio Pagnotta, Briland Hitaj, Luigi V. Mancini, and Fernando Pérez-Cruz. 2022. MaleficNet: Hiding Malware into Deep Neural Networks Using Spread-Spectrum Channel Coding. In *European Symposium on Research in Computer Security.* 425–444.

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of the 10th International Conference on Learning Representations.* 1–13.

[13] Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards Reasoning in Large Language Models: A Survey. In *Findings of the Association for Computational Linguistics: ACL 2023.* 1049–1065.

[14] Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. 2024. Catastrophic Jailbreak of Open-source LLMs via Exploiting Generation. In *Proceedings of the 12th International Conference on Learning Representations.* 1–21.

[15] Jabari Kwesi, Jiaxun Cao, Riya Manchanda, and Pardis Emami-Naeini. 2025. Exploring user security and privacy attitudes and concerns toward the use of

---

[7]https://protectai.com/blog/why-ebpf-is-secure.

General-Purpose LLM chatbots for mental health. In *Proceedings of the 34th USENIX Security Symposium*. 6007–6024.

[16] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 16: 611–626.

[17] Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. 2024. BioMistral: A Collection of Open-Source Pretrained Large Language Models for Medical Domains. In *Findings of the Association for Computational Linguistics: ACL 2024*. 5848–5864.

[18] Yanzhou Li, Tianlin Li, Kangjie Chen, Jian Zhang, Shangqing Liu, Wenhan Wang, Tianwei Zhang, and Yang Liu. 2024. BadEdit: Backdooring Large Language Models by Model Editing. In *Proceedings of the 12th International Conference on Learning Representations*. 1–18.

[19] Tao Liu, Zihao Liu, Qi Liu, Wujie Wen, Wenyao Xu, and Ming Li. 2020. StegoNet: Turn Deep Neural Network into a Stegomalware. In *Proceedings of the 36th Annual Computer Security Applications Conference*. 928–938.

[20] Tong Liu, Guozhu Meng, Peng Zhou, Zizhuang Deng, Shuaiyin Yao, and Kai Chen. 2025. The Art of Hide and Seek: Making Pickle-Based Model Supply Chain Poisoning Stealthy Again. arXiv:2508.19774 [cs.CR] https://arxiv.org/abs/2508.19774

[21] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. 61–68.

[22] Mmaitre314. 2024. Python pickle malware scanner. https://pypi.org/project/picklescan/.

[23] Fengran Mo, Chuan Meng, Mohammad Aliannejadi, and Jian-Yun Nie. 2025. Conversational search: From fundamentals to frontiers in the LLM era. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 4094–4097.

[24] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-Free Quantization Through Weight Equalization and Bias Correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1325–1334.

[25] OpenAI. 2023. *GPT-4 Technical Report*. Technical Report arXiv:2303.08774.

[26] ProtectAI. 2024. Modelscan: Protection against model serialization attacks. https://github.com/protectai/modelscan.

[27] Python Software Foundation. 2025. pickle — Python object serialization.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI Technical Report* 1, 8 (2019), 9.

[29] Tianyi Tang, Wenyang Luo, Haoyang Huang, Dongdong Zhang, Xiaolei Wang, Wayne Xin Zhao, Furu Wei, and Ji-Rong Wen. 2024. Language-Specific Neurons: The Key to Multilingual Capabilities in Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. 5701–5715.

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* abs/2302.13971 (2023), 1–27.

[31] Yu-Lin Tsai, Chia-Yi Hsu, Chia-Mu Yu, and Pin-Yu Chen. 2021. Formalizing generalization and adversarial robustness of neural networks to weight perturbations. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*. 13 pages.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5998–6008.

[33] Zhi Wang, Chaoge Liu, and Xiang Cui. 2021. EvilModel: Hiding Malware Inside of Neural Network Models. In *2021 IEEE Symposium on Computers and Communications*. 1–7.

[34] Zhi Wang, Chaoge Liu, Xiang Cui, Jie Yin, and Xutong Wang. 2022. EvilModel 2.0: Bringing Neural Network Models into Malware Attacks. *Computers & Security* 120 (2022), 102807.

[35] Yu Xia, Subhojyoti Mukherjee, Zhouhang Xie, Junda Wu, Xintong Li, Ryan Aponte, Hanjia Lyu, Joe Barrow, Hongjie Chen, Franck Dernoncourt, Branislav Kveton, Tong Yu, Ruiyi Zhang, Jiuxiang Gu, Nesreen K. Ahmed, Yu Wang, Xiang Chen, Hanieh Deilamsalehy, Sungchul Kim, Zhengmian Hu, Yue Zhao, Nedim Lipka, Seunghyun Yoon, Ting-Hao 'Kenneth' Huang, Zichao Wang, Puneet Mathur, Soumyabrata Pal, Koyel Mukherjee, Zhehao Zhang, Namyong Park, Thien Huu Nguyen, Jiebo Luo, Ryan A. Rossi, and Julian McAuley. 2025. From Selection to Generation: A Survey of LLM-based Active Learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. 14552–14569.

[36] Jianhan Xu, Linyang Li, Jiping Zhang, Xiaoqing Zheng, Kai-Wei Chang, Cho-Jui Hsieh, and Xuanjing Huang. 2022. Weight Perturbation as Defense against Adversarial Word Substitutions. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. 7054–7063.

[37] Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. 2024. LLM-Fuzzer: Scaling Assessment of Large Language Model Jailbreaks. In *Proceedings of the 33th USENIX Security Symposium*. 4657–4674.

[38] Hangfan Zhang, Zhimeng Guo, Huaisheng Zhu, Bochuan Cao, Lu Lin, Jinyuan Jia, Jinghui Chen, and Dinghao Wu. 2024. Jailbreak Open-Sourced Large Language Models via Enforced Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. 5475–5493.

[39] Rui Zhang, Hongwei Li, Rui Wen, Wenbo Jiang, Yuan Zhang, Michael Backes, Yun Shen, and Yang Zhang. 2024. Instruction Backdoor Attacks Against Customized LLMs. In *Proceedings of the 33th USENIX Security Symposium*. 1849–1866.

[40] Rui Zhang, Hongwei Li, Rui Wen, Wenbo Jiang, Yuan Zhang, Michael Backes, Yun Shen, and Yang Zhang. 2025. JBShield: Defending Large Language Models from Jailbreak Attacks through Activated Concept Analysis and Manipulation. In *Proceedings of the 34th USENIX Security Symposium*. 1–23.

[41] Zhihao Zhang, Jun Zhao, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. Unveiling Linguistic Regions in Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. 6228–6247.

[42] Na Zhao, Kejiang Chen, Chuan Qin, Yi Yin, Weiming Zhang, and Nenghai Yu. 2023. Calibration-based Steganalysis for Neural Network Steganography. In *Proceedings of the 2023 ACM Workshop on Information Hiding and Multimedia Security*. 91–96.

[43] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Mexico City, Mexico, 2299–2314. doi:10.18653/v1/2024.findings-naacl.149

[44] Ruofan Zhu, Ganhao Chen, Wenbo Shen, Xiaofei Xie, and Rui Chang. 2025. My Model is Malware to You: Transforming AI Models into Malware by Abusing TensorFlow APIs. In *2025 IEEE Symposium on Security and Privacy*. 486–503.
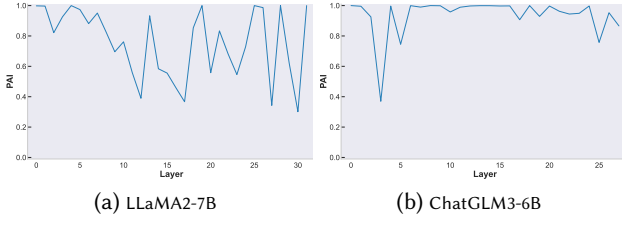
(a) LLaMA2-7B

(b) ChatGLM3-6B

Figure 9: d_{PAI} **of models with** $n = 4$**. Results are averaged over 3 runs with different random seeds on MMLU**

(a) D_{acc} of LLaMA2-7B

(b) D_{acc} of ChatGLM3-6B

(c) D_{ppl} of LLaMA2-7B

(d) D_{ppl} of ChatGLM3-6B

Figure 10: D_{acc} **&** D_{ppl} **of models with** n**=4. Results are averaged over 3 runs with different random seeds on MMLU.**

(a) D_{acc} of LLaMA2-7B

(b) D_{acc} of ChatGLM3-6B

(c) D_{ppl} of LLaMA2-7B
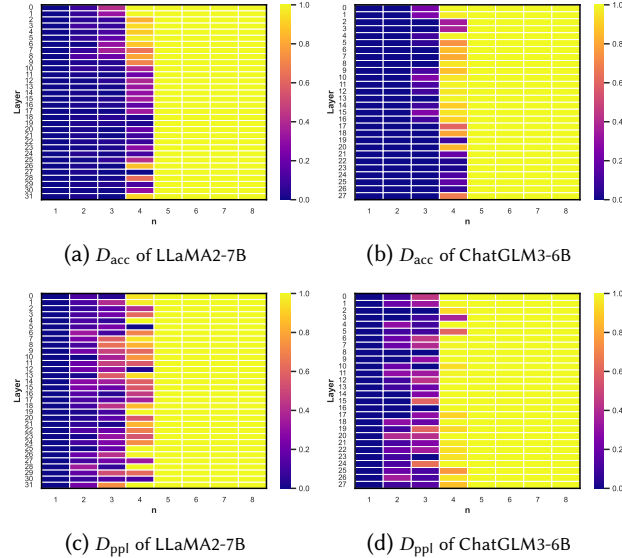
(d) D_{ppl} of ChatGLM3-6B

Figure 11: D_{acc} **&** D_{ppl} **of LLaMA2-7B and ChatGLM3-6B on MMLU in the case of 8-bit quantization. Results are averaged over 3 runs with different random seeds.**

A   Appendix: Results of SASER with *robust model*

We plot the attack performance in the setting of 8-bit quantization.

Grouping methods of targeted parameters. As shown in Figure 7, d_{PAI} increases along with the increase of n from 0 to 1 across all grouping methods, which is in line with the results in the *general mode*.
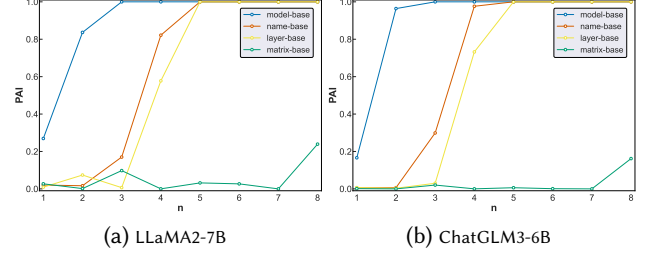

(a) LLaMA2-7B

(b) ChatGLM3-6B

Figure 7: d_{PAI} **of grouping methods with** n **ranging from 1 to 8. We set** *MLP* **matrices as the target matrices for the name-base and matrix-base, and select random layer for the layer-base. Results are averaged over 3 runs with different random seeds.**

LSB numbers. Figure 8 shows d_{PAI} of LLaMA2-7B and ChatGLM3-6B with n ranging from 1 to 8. For most layers, d_{PAI} decreases sharply when $n = 4$. In this case, payloads with a size no more than 37.5% of the model size can be embedded without detection.
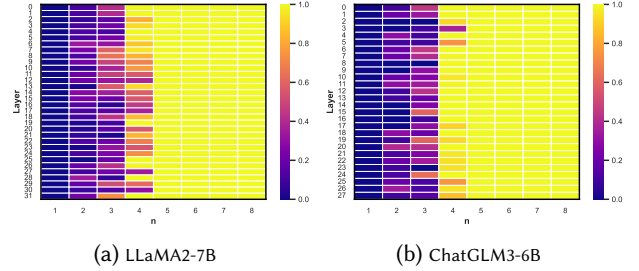

(a) LLaMA2-7B

(b) ChatGLM3-6B

Figure 8: d_{PAI} **of models on MMLU with 8-bit quantization. Results are averaged over 3 runs with different random seeds.**

Distribution of vulnerable parameters. Figure 9 illustrates more perturbation and uncertainty of d_{PAI} compared to the *general mode*. The reason behind this observation is that integer values have more variability than floating-point values under LSB embedding methods with the same random bits, impacting model performance to a larger extent.

Contributions of D_{acc} **and** D_{ppl}**.** Similar results and findings can be obtained by aligning Figure 10 & 5, and Figure 11 & 6.