CoreGuard: Safeguarding Foundational Capabilities of LLMs Against Model Stealing in Edge Deployment

Qinfeng Li[1] **Tianyue Luo**[1,2] **Xuhong Zhang**[1,3*] **Yangfan Xie**[1] **Zhiqiang Shen**[1]
Lijun Zhang[4] **Yier Jin**[5] **Hao Peng**[6] **Xinkui Zhao**[1] **Xianwei Zhu**[7] **Jianwei Yin**[1]

[1]School of Software Technology, Zhejiang University
[2]Institute of Software, Chinese Academy of Sciences
[3]Ningbo Global Innovation Center, Zhejiang University
[4]Washington University in St. Louis
[5]University of Science and Technology of China
[6]College of Computer Science and Technology, Zhejiang Normal University
[7]China Electronics Technology Design and Research Institute

Abstract

Proprietary large language models (LLMs) exhibit strong generalization capabilities across diverse tasks and are increasingly deployed on edge devices for efficiency and privacy reasons. However, deploying proprietary LLMs at the edge without adequate protection introduces critical security threats. Attackers can extract model weights and architectures, enabling unauthorized copying and misuse. Even when protective measures prevent full extraction of model weights, attackers may still perform advanced attacks, such as fine-tuning, to further exploit the model. Existing defenses against these threats typically incur significant computational and communication overhead, making them impractical for edge deployment. To safeguard the edge-deployed LLMs, we introduce CoreGuard, a computation- and communication-efficient protection method. CoreGuard employs an efficient protection protocol to reduce computational overhead and minimize communication overhead via a propagation protocol. Extensive experiments show that CoreGuard achieves upper-bound security protection with negligible overhead.

1 Introduction

Large language models (LLMs), especially proprietary ones, such as ChatGPT [28] and Claude [4], demonstrate exceptional generalization ability across various tasks [6, 31]. Additionally, deploying LLMs on edge devices is a growing trend for latency- and privacy-sensitive tasks, e.g., Apple Inc. introduced Apple Intelligence, which integrates a 3-billion-parameter LLM into users' devices in the latest iOS version [5]. However, when these proprietary LLMs are deployed to edge devices without adequate protection, adversaries can extract detailed model information (including architecture and weights) through software analysis techniques [7, 45], leading to unauthorized copying and misuse outside the intended device. Even if some protections prevent attackers from fully extracting the original weights, attackers can still perform more advanced attacks, such as fine-tuning the partially recovered model to exploit its embedded knowledge and strong generalization capabilities for new tasks. We refer to this threat as *foundational capability stealing*. This threat is especially practical for proprietary, domain-specific LLMs trained on private data, like BloombergGPT [46] in finance or Med-PaLM 2 [36] in healthcare, where comparable open-source alternatives are limited. Considering the substantial resources required to develop high-performance LLMs [41], it is crucial to ensure robust protection against these threats in edge deployments.

* Corresponding to zhangxuhong@zju.edu.cn

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

 https://doi.org/10.52202/085713-2327

Table 1: Comparison with existing solutions. ✓/✗ indicate whether each property is satisfied.

Solutions (exemplar)	Proactivity	Runtime Security	Backbone Protection	**Sufficiency**	**Efficiency**
Watermarking [1]	✗	✗	✗	✗	✓
Model encryption [52]	✓	✗	✗	✗	✓
TPTE [51]	✓	✓	✗	✗	✗
PPTE [24]	✓	✓	✓	✗	✗
PSP [38]	✓	✓	✓	✓	✗
CoreGuard (ours)	✓	✓	✓	✓	✓

Unfortunately, as shown in Table 1, traditional solutions struggle to protect the edge-deployed LLMs. Specifically, passive protection methods, such as watermark [1, 15, 21], are not applicable since only the proof of ownership is insufficient in such an unsupervised edge operation scenario, where attackers can misuse the model without detection. In contrast, active protection works by allowing only authorized users to use the well-performed model. For example, some work encrypts models before deploying them on devices [52, 22], and these models are only decrypted before execution. However, it's crucial to recognize that while these solutions can implement effective protection before the inference state, current studies [7, 45] suggest that, even after decryption, models remain susceptible to runtime attacks during inference, i.e., attackers reverse engineer the models in their runtime state.

To defend against runtime attacks, one potential solution [25, 8] is to place the model into a secure execution environment, e.g., a trusted execution environment (TEE), which are typically implemented as a CPU-based enclave (e.g., ARM TrustZone, Intel SGX) that stores sensitive data and safeguards against runtime attacks. However, directly placing the entire model within a TEE is impractical, as it results in approximately a 50× reduction in model efficiency due to the TEE's limited computational speed [43]. Thus, some researchers propose only putting the most critical parameters in TEEs and offloading the rest computation to GPUs. For example, Zhang et al. [51] protect only task-related adapters within TEE and offload the model backbone to GPUs. However, such approaches are primarily effective for traditional task-specific models and are insufficient for protecting LLMs, as they leave the model backbone directly exposed.

To protect the backbone, a straightforward idea [24, 35] is to place a subset of the backbone (e.g., the last layer) in the TEE, i.e., Partial Parameter TEE Execution (**PPTE**). However, prior work [51] shows that PPTE only provides insufficient protection, and this limitation becomes even more critical when applied to LLMs. Specifically, PPTEs crudely execute weights in TEE for protection, where the limited computational power of TEEs restricts the number of protected weights. The scarcity of protected weights makes it easy for attackers to reconstruct them, even with just 1% of the training datasets, compromising security [51]. Even worse, if an attacker aims to exploit a LLM's foundational capabilities, their target task is likely one for which they already have abundant labeled data (e.g., 100% training set), making theft easier.

To increase the number of protected weights, a promising approach is to protect weights through shuffling, i.e., Parameter Shuffling Protection (**PSP**) [38, 20]. For example, ShadowNet [38] protects model weights by shuffling the channels of convolutional kernels. The protection ensures that only the corresponding shuffled input can be correctly computed with the shuffled weights. This input-shuffling process is performed within the TEE, thus ensuring its security. However, the excessive data transfer overhead between the TEE and GPU makes ShadowNet impractical for LLMs. Specifically, each shuffled layer requires transferring its input from the GPU to the TEE and back, resulting in 448 TEE-CPU transfers for a LLaMA3-8B model with 224 linear layers (each linear layer requires 2 transfers) to generate a single token. Therefore, with an input of 128 context length, each transfer would average 3MB of data (assuming float-32 precision), leading to a total data transfer volume of approximately 1.3GB (448 × 3MB). Given that mainstream mobile platform TEEs, like TrustZone, have a transfer rate of about 1GB/s between TEE and GPU [3], generating a single token takes about 1.3 seconds. Consequently, producing a complete output would require several hundred seconds (assuming it consists of 100 tokens) solely for data TEE-GPU transfer.

In summary, maintaining acceptable computation and communication overhead under sufficient security of LLM in edge deployment is an unresolved challenge for existing solutions. To address this, we propose CoreGuard, a computation- and communication-efficient approach designed to prevent the model from working without the proper authorization from the trusted hardware, i.e., TEE, within the edge device. To reduce TEE execution overhead, CoreGuard is inspired by prior

PSP solutions by securing parameters through obfuscation, which allows model computations to be performed on the GPU. Specifically, it employs a *protection protocol* that row-permutes the weight matrices of linear layers, ensuring their input features must be correspondingly column-permuted (i.e., authorization) by TEE. Crucially, to avoid requiring TEE authorization for each linear layer and minimize TEE-GPU transfer overhead, CoreGuard introduces a *propagation protocol* that reduces TEE authorizations to a single initial authorization. After this, all subsequent protected layers apply column permutations to their outputs, enabling the initial authorization to be propagated.

Our evaluation shows that CoreGuard outperforms existing defenses in security and efficiency. Besides, the experimental results show no difference in accuracy between the CoreGuard-protected model and the original model. The contributions of this work are as follows:

- We are the first to address the protection of foundational capabilities in edge-deployed LLMs. Our work systematically characterizes the security challenges in this setting and identifies the requirements for the protection of edge-deployed LLMs.
- We propose CoreGuard, a plug-and-play solution that utilizes a lightweight authorization mechanism to protect edge-deployed LLMs. It employs a propagation protocol, significantly reducing transfer overhead while maintaining a low computation overhead.
- Extensive experiments demonstrate that compared to the existing solutions, CoreGuard offers a higher security guarantee with lower overhead and no accuracy loss.

2 Threat Model

In this paper, we consider two parties: the defender and the attacker. The defender is the party that owns the edge-deployed model. The attacker aims to steal the model.

Defender's Goal. The defender aims to deploy a locked model on the device, ensuring it works only with proper authorization from the trusted hardware (i.e., TEE) within the device. The defender can control its model and modify it to ensure protection. This protection ensures that, when correctly authorized, the model permits normal queries from authorized users, while other attacks based on these legitimate queries (e.g., distillation attacks) are orthogonal to our work.

Adversary's Goal. The attacker aims to abuse the deployed model off-device (i.e., without TEE authorization) for their task. A straightforward way is to try to reverse the authorization process so that the locked model can be used independently of the device. Another more practical new way is to fine-tune the locked model to obtain a model that excels at a desired task [20, 51].

Adversary's Capability. The attacker could ***decide the target task*** and ***possess sufficient well-labeled data***, whereas prior work often assumes access to only 1% of the dataset [51]. In this paper, we consider TEE to be a secure world, while other hardware (e.g., GPU and CPU) could be white-box exposed to attackers. Therefore, the attacker can have access to the details, e.g., model architecture and weights, of the locked model outside TEE.

3 Design of CoreGuard

This section presents our proposed protection method, CoreGuard, which utilizes a permutation strategy to address the key requirements outlined in Section 1.

3.1 Approach Overview

As shown in Figure 1, CoreGuard operates in two phases: model locking (before deployment) and inference authorization (post-deployment). In the *model locking phase*, CoreGuard locks a trained model by applying a **protection protocol** to the weights of linear layers, i.e., swapping rows of the weight matrix. These row permutations act as a *lock*, rendering the linear layers dysfunctional, thus making the overall model unusable. These locked layers can only function properly with inputs that are correspondingly column-permuted, which essentially acts as *authorization*. However, directly using a TEE to authorize each locked layer would result in significant TEE-GPU transfer overhead. To address this, CoreGuard proposes a **propagation protocol**, which enables the features to be column-permuted by the network itself. Specifically, CoreGuard permutes the columns of certain

 https://doi.org/10.52202/085713-2327

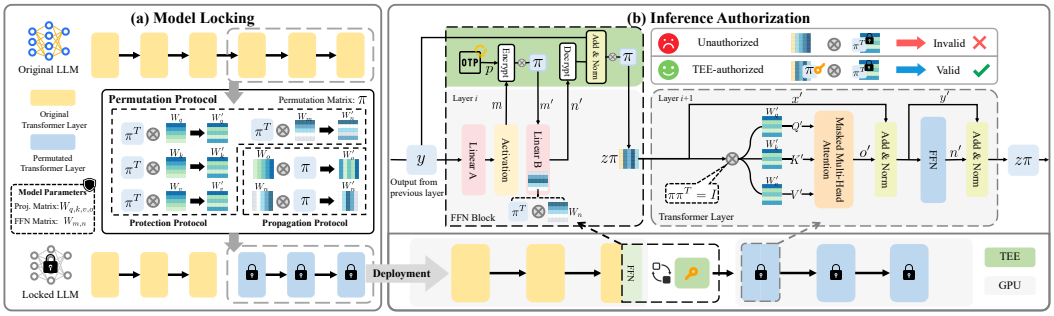

Figure 1: An overview of CoreGuard. (a) Model locking: before deployment, CoreGuard permutes layers in the original model, thus creating a locked model. (b) Inference authorization: during inference, the input feature of the permuted layers is authorized, which is integrated within the FFN block of the preceding transformer layer.

layers, thus through these layers' operation, their output features are column-permuted, which achieve authorization similarly. In this way, the TEE only needs to manage the initial authorization, and the authorization can be propagated to all subsequent layers.

The *inference authorization phase*, as shown in the black dashed box of Figure 1 (b), aims to securely perform initial authorization. A naive method directly authorizes the original output z in the TEE and returns $z\pi$, but this obviously will leak π via input-output comparisons. One solution is to place both Linear B and the add-norm layer inside the TEE, fully hiding z, but executing Linear B in the TEE introduces high overhead. To reduce cost, we offload linear B to the GPU. However, exposing Linear B's output n will leak z. Therefore, we apply OTP noise p to Linear B's input m. Since OTP also requires protection, we apply a permutation after encryption, producing m' (i.e., $(m+p)\pi$). Correspondingly, Linear B's weights are pre-permuted by π^T to offset π. Overall, as shown in the Figure 1 (b), the feature enters the TEE twice: first for OTP encryption (encrypting m to m'), and second for performing authorization to produce $z\pi$.

3.2 Model Locking

Given a classic transformer model, we describe how to lock a transformer layer within it. We first apply the protection protocol to layers involved in input feature projection (e.g., the QKV projection layer) to secure the model. Then, the propagation protocol is applied to layers managing output projection (e.g., the output projection layer in the attention block and the FFN block). Finally, we demonstrate the functionality of the locked transformer layer.

Transformer Layer Formalization. We begin by formalizing a standard transformer layer. Let x, and $z \in \mathbb{R}^{l\times d}$ denote its input and output, where l is the sequence length (e.g., the number of tokens) and d is the model dimension. We define a classic transformer layer as a function $f_w : \mathbb{R}^{l\times d} \to \mathbb{R}^{l\times d}$ with weight parameters w. The transformer layer, i.e., $f_w(x) = z$, is computed as follows:

$$
\begin{aligned}
Q = xW_q, K = xW_k, V = xW_v, &\quad //QKV\ project \\
o = \text{softmax}\left(\frac{QK^T}{\sqrt{d/h}} + M\right)VW_o, &\quad //Attention \\
y = \gamma_1 \odot \frac{o + x - \mu_{o+x}}{\sigma_{o+x}} + \beta_1, &\quad //Add\ \&\ norm \\
m = \text{ReLU}(yW_m + b_m), &\quad //FFN\ input \\
n = mW_n + b_n &\quad //FFN\ output \\
z = \gamma_2 \odot \frac{y + n - \mu_{y+n}}{\sigma_{y+n}} + \beta_2, &\quad //Add\ \&\ norm
\end{aligned}
\tag{1}
$$

where w includes the attention weights W_q, W_k, W_v, and $W_o \in \mathbb{R}^{d\times d}$, the add-norm weights γ_1, β_1, γ_2, and $\beta_2 \in \mathbb{R}^d$, the FFN weights W_m and $W_n \in \mathbb{R}^{d\times d}$, the bias b_m and $b_n \in \mathbb{R}^d$. h is the number of attention heads. The mask $M \in \mathbb{R}^{d\times d}$ is an all-zero matrix in the encoder and has negative infinity in its upper right corner in the decoder. Notably, among these layers, we refer to W_q, W_k, and W_v in the attention block, and W_m in the FFN block as *input-processing layers* as they process

the input feature of their blocks. We refer to W_o, W_n, and the add-norm weights in each block as *output-processing layers* as they manage the output of the blocks.

Protection Protocol. To achieve protection, we row-permutate the *input-processing layers* for protection. These layers process the module's inputs directly, thus they have the ability to cause computation failures if the input is not authorized, leading to incorrect results.

Specifically, let $\pi \in \{0,1\}^{d\times d}$ denote a permutation matrix, where $\forall \pi, \pi\pi^T = I$, where I is the identity matrix, a property of the permutation matrix. We row-permute the parameters w:

$$W'_q = \pi^T W_q, \quad W'_k = \pi^T W_k, \quad W'_v = \pi^T W_v, \quad W'_m = \pi^T W_m. \tag{2}$$

Thus, only the corresponding column-permuted input can be computed with these layers. For instance:

$$x\pi\pi^T W_q = xW_q = Q, \tag{3}$$

where input's column permutation (i.e., π, the authorization) and row permutations (i.e., π^T, the lock) offset each other ($\pi\pi^T = I$), resulting in the same computation as the original.

Propagation Protocol. The propagation protocol aims to avoid repeated TEE authorization by allowing each transformer layer to automatically receive an authorized input $z\pi$ from the previous layer. Specifically, the output of a transformer layer is directly determined by four output-processing layers: W_o, W_n, and two add-norm layers. Once all their outputs are column-permuted, the overall output is column-permuted, thus achieving automatic authorization. Therefore, the problem simplifies to column-permuting the output of a single layer. For instance, as illustrated below, column-permuting W_n to $W_n\pi$ transforms its original output n into a column-permuted output $n\pi$:

$$n' = mW'_n + b'_n = mW_n\pi + b_n\pi = n\pi, \tag{4}$$

Therefore, to implement propagation protocol, we column-permutate all *output-processing layers*, ensuring the feature can be re-authorized before exiting the module:

$$W'_o = W_o\pi, \quad \gamma'_1 = \gamma_1\pi, \quad \beta'_1 = \beta_1\pi, \quad W'_n = W_n\pi, \quad b'_n = b_n\pi, \quad \gamma'_2 = \gamma_2\pi, \quad \beta'_2 = \beta_2\pi. \tag{5}$$

Locked Transformer Layer Formalization. With the permuted weights (denoted as w'), taking $x\pi$ as authorized input, its functionality can be described as follows:

$$\begin{aligned}
Q' &= x\pi\pi^T W_q = xW_q = Q,\\
K' &= x\pi\pi^T W_k = xW_k = K,\\
V' &= x\pi\pi^T W_v = xW_v = V,\\
o' &= \text{softmax}\left(\frac{Q'K'^T}{\sqrt{d/h}} + M\right)V'W_o\pi = o\pi,\\
y' &= \gamma_1\pi \odot \frac{o' + x\pi - \mu_{x\pi+o'}}{\sigma_{x\pi+o'}} + \beta_1\pi = y\pi.\\
m' &= \text{ReLU}(y'\pi^T W_m + b_m) = m.\\
n' &= m'W_n\pi + b_n\pi = n\pi,\\
z' &= \gamma_2\pi \odot \frac{n' + y' - \mu_{y'+n'}}{\sigma_{y'+n'}} + \beta_2\pi = z\pi.
\end{aligned} \tag{6}$$

Thus, the functionality of the locked layer can be represented as $f_{w'}(x') = z\pi = f_w(x)\pi$, valid only when $x' = x\pi$, thereby preventing unauthorized access (without π). When the permuted transformer layer receives authorized input ($x\pi$), its output ($z\pi$) matches the original output (z) with a column permutation, authorizing the next layer. This propagation is consistent across all subsequent layers. Therefore, authorization is required only for the first permuted layer.

3.3 Inference Authorization

During the inference, the initial column-permutated feature ($x\pi$) is generated while ensuring the security of this authorization process. As shown in Figure 1, TEE encrypts the feature before the FFN block's output linear layer. After the GPU processes the feature with this layer, it re-enters the

 https://doi.org/10.52202/085713-2327

TEE for decryption. Finally, the add-norm computation is performed, followed by the permutation of the output before it is returned. The detailed descriptions are provided in the following subsections.

Encryption. At the beginning, the input linear layer of the FFN block receives y as input from the previous layer:

$$m = \text{ReLU}(yW_m + b_m). \tag{7}$$

Following this, before the output linear layer, the feature is encrypted using an OTP. Additionally, to protect this encryption process from input-output differencing, we introduce positional obfuscation by applying a permutation:

$$m' = m\pi + p\pi, \tag{8}$$

where p is the pad, a noise matrix of the same shape as m, and m' is the encrypted permuted feature. The principle of the one-time pad (OTP) [34] ensures that p is different each time, thus even for the same m, m' produced will differ. In this way, the OTP encryption (i.e., p) and the permutation (i.e., π) can conceal each other.

Output Linear Layer. The FFN block's output linear layer processes the encrypted feature. However, since m' is permuted, the layer's parameters must be pre-aligned to ensure correct computations. Specifically, we correspondingly permute the output linear layer (W_n) *before deployment*:

$$W'_n = \pi^T W_n. \tag{9}$$

With the above preparation, *during inference*, the encrypted feature m' is transferred to GPUs and processed by the permuted output linear layer W'_n:

$$n' = m'W'_n = (m\pi + p\pi)\pi^T W_n + b_n = n + pW_n. \tag{10}$$

Decryption. If the OTP noise (i.e., pW_n) is not eliminated, the network's functionality is compromised. Specifically, OTP implementation must meet two requirements: 1) conceal both the encryption and decryption; 2) all computations on the feature before decryption are linear. To meet these requirements, n' is transferred to the TEE for decryption:

$$n'' = n' - pW_n = n. \tag{11}$$

Notably, following prior work [43], pW_n can be conducted by the model provider or in an offline phase. Both strategies do not increase the overhead of online inference or impede its efficiency [51].

Authorization. Lastly, the decrypted feature is processed by the add-norm layer and permuted to achieve authorization in TEE. The steps are as follows:

$$z' = (\gamma_2 \odot \frac{n + y - \mu_{y+n}}{\sigma_{y+n}} + \beta_2)\pi = z\pi, \tag{12}$$

where z' (i.e., $z\pi$) is the authorized feature, which will be the input feature for the subsequent permuted transformer layer, thus achieving authorized usage. Notably, the TEE *only authorizes once for each inference*, minimizing communication overhead by limiting TEE-GPU transfers to 5 rounds. Furthermore, it uses only lightweight computations (e.g., matrix addition), ensuring minimal computation overhead.

Authorization Position. The TEE authorization position, which determines the number of layers to be locked, is a hyperparameter. CoreGuard sets this position to the midpoint to enhance security, as detailed in Appendix B. Specifically, when the authorization point is in the middle, attackers must recover more parameters, increasing the difficulty of theft. Essentially, permutation only maps parameters to a new domain without disrupting their functionality, meaning the attacker only needs to recover the missing layers, whether in the original or the new domain, to obtain a complete model. If the authorization occurs at the beginning or the end, attackers can retrain just one layer to recover, which is similar to prompt tuning or training a classification head. In contrast, placing it in the middle requires the attacker to restore at least half of the parameters, which is more difficult.

Security Analysis. Potential attackers might attempt to steal the locked model by recovering the permuted parameters. However, it is impossible as the probability of guessing the correct π is $1/(d!)$. In practice, d is typically larger than 512, e.g., $d = 4096$ in LLaMA3-8B [42].

Alternatively, attackers may attempt to recover π by exploiting the TEE's functionality—for instance, by trying to crack π from the TEE's inputs and outputs. However, accurately solving π is infeasible, as the problem is ill-posed. Specifically, even with auxiliary information, the task reduces to solving a Learning With Errors (LWE) problem, which is widely regarded as NP-hard (see Appendix C).

Although solving π exactly is impossible, attackers might try to approximate TEE's functionality to facilitate model stealing. For example, trying to learn a mapping from y to $z\pi$ to bypass OTP encryption. However, this approach is also ineffective (as shown in Appendix D): the mapping is nonlinear, involves a massive number of parameters, and even minor approximation errors can invalidate the result [11].

4 Experiments

In this section, we perform extensive experiments to answer the following research questions (**RQ**s):

RQ1: How secure is CoreGuard, and does it effectively protect the foundational capabilities of LLMs? **RQ2:** How does CoreGuard's computation and communication overhead compare to other defenses? **RQ3:** Does CoreGuard sacrifice the accuracy of the model?

4.1 Experimental Settings

Datasets. To evaluate CoreGuard, we assume the attacker attempts to steal the LLM to different target tasks, including four domain-specific tasks: GSM8k (mathematics) [9], Spider (code generation) [50], PubMedQA (medical question answering) [16], and SQuAD (reading comprehension) [32].

Models. We choose four representative LLMs for validation. Two of them are specifically designed for on-device deployment: Qwen2-0.5B-Instruct [49], Gemma2-2B-it [40]. The other two are larger models: ChatGLM3-6B-32k [11] and LLaMA3-8B-Instruct [42].

Metric. For all tasks, we use accuracy as the metric. Specifically, for GSM8k, a prediction is considered correct when the final answer matches the actual result. For PubMedQA, a 3-class classification task is deemed correct if it matches the true label. For Spider, we follow the prior work [19] and assess whether the generated query matches the reference query. For SQuAD, we align the answer with the reference, as in previous work [48]. To evaluate execution overhead, we use Floating Point Operations (FLOPs) as the metric, following the prior research [39, 14].

Implementation Details. We conduct experiments with the *Huggingface library* [13]. For optimization, we use the widely adopted AdamW [17] optimizer and a linear learning rate scheduler [47]. Same as previous work [47], we report results of the runs that achieve the highest performance, consistent with real-world practices prioritizing the optimal model. All experiments are conducted on NVIDIA A800 GPUs with 80GB of VRAM.

Baselines. We compare our method against comprehensive baselines, including ideal bounds and representative defenses, categorized by their protection principles (TPTE, PPTE, PSP): ① **Lower bound**: (i) *No-shield*, where the adversary accesses the unprotected model directly. ② **Upper bound**: (i) *Black-box*, where only model architecture is visible to the attacker, offering the strongest protection. ③ **TPTE** (task-only protection): (i) NPLO [51]. ④ **PPTE** (partial model protection): (i) DarkneTZ [24]; (ii) SOTER [35]; (iii) Serdab [12]; (iv) Our baseline, DTE, runs the latter transformer layers within TEE. ⑤ **PSP** (parameter shuffling): (i) ShadowNet [38]; (ii) TransLinkGuard (TLG) [20]. To adapt these methods to transformer models, we rigorously configure each solution based on its papers. Specifically, for SOTER, TEE randomly shields 20% layers. For Serdab, the TEE shields the first transformer layer. ShadowNet obfuscates all the linear transformation layers. For DarkneTZ, the last transformer layer is put into TEE.

Model Stealing Attack. As discussed in prior work [51], we identify finetuning attacks as a major threat to edge-protection solutions. The finetuning attack, as detailed in Appendix A, is well-defined in prior studies [51, 20], and involves two main steps. First, the attacker builds a surrogate model using available parameters of the target model from unsecured environments. Second, they train this surrogate model with accessible datasets to perform the target task. Notably, due to the generalization ability of LLMs, attackers can steal the model for any target task, where they may possess sufficient training datasets to achieve stealing. Therefore, we assume the attacker has the entire training dataset (100%), a more stringent condition than the 1% dataset employed in previous research.

 https://doi.org/10.52202/085713-2327

Table 2: Security assessment of CoreGuard in preventing unauthorized direct inference and model stealing attack. For *direct inference*, we report the authorized ("Auth") and unauthorized ("Unau") usage accuracies (%). For *model stealing attacks*, we report the attack accuracies (%), *lower attack accuracies indicate stronger defense*. The last row reports the average attack accuracies of each defense relative to the baseline black-box solutions. The column representing CoreGuard ("Ours") is highlighted in **bold**.

		Direct Inference		**Model Stealing Attack ↓**										
		Unau ↓	Auth	No-shield	NPLO	Serdab	DarkneTZ	SOTER	TLG	ShadowNet	DTE	**Ours**	Black-box
Qwen2-0.5B	GSM8k	0.00±0.00	15.51±1.02	21.53±1.43	20.92±1.21	14.96±0.88	16.81±1.07	12.50±0.91	1.43±0.04	1.34±0.04	2.36±0.06	**2.41**±0.07	1.29±0.03
	Spider	0.00±0.00	5.56±0.62	28.48±1.54	30.28±1.73	23.90±1.21	26.01±1.47	21.52±1.03	3.31±0.10	3.67±0.11	3.92±0.12	**3.79**±0.11	3.81±0.11
	PubMedQA	0.00±0.00	15.50±1.24	58.00±2.56	56.50±2.47	49.00±2.02	51.50±2.19	47.00±1.94	3.50±0.14	4.50±0.18	5.50±0.22	**6.00**±0.24	5.00±0.20
	SQuAD	0.00±0.00	16.50±1.28	30.54±1.75	32.33±1.89	28.42±1.53	29.89±1.64	26.34±1.47	6.81±0.27	5.93±0.24	4.42±0.18	**7.35**±0.29	5.66±0.23
Gemma2-2B	GSM8k	0.00±0.00	30.10±2.05	40.94±2.57	40.50±2.41	35.18±1.96	37.07±2.10	32.67±1.72	4.58±0.18	10.81±0.43	4.56±0.18	**3.91**±0.16	1.74±0.07
	Spider	0.00±0.00	3.52±0.38	39.15±1.71	38.83±1.65	24.80±1.08	23.29±0.98	12.81±0.63	0.00±0.00	0.00±0.00	0.00±0.00	**0.00**±0.00	0.00±0.00
	PubMedQA	0.00±0.00	10.50±0.83	69.50±3.21	69.00±3.12	55.50±2.41	60.00±2.63	55.50±2.32	10.50±0.42	7.00±0.28	9.50±0.38	**12.00**±0.48	6.50±0.26
	SQuAD	0.00±0.00	43.21±2.76	63.96±3.08	63.94±3.07	60.82±2.81	61.02±2.84	57.87±2.63	7.91±0.32	6.71±0.27	7.51±0.30	**7.81**±0.31	8.81±0.35
ChatGLM3-6B	GSM8k	0.00±0.00	37.13±2.33	55.95±2.87	55.07±2.74	53.67±2.58	54.91±2.74	54.55±2.71	2.84±0.11	0.43±0.02	0.93±0.04	**1.04**±0.04	0.23±0.01
	Spider	0.00±0.00	5.15±0.61	35.81±1.94	37.03±2.03	32.25±1.64	33.81±1.79	33.22±1.75	6.19±0.25	8.31±0.33	8.44±0.34	**7.37**±0.29	7.91±0.32
	PubMedQA	0.00±0.00	46.00±3.12	71.00±3.34	70.00±3.21	63.00±2.89	65.50±3.04	60.50±2.63	10.00±0.40	12.00±0.48	12.00±0.48	**12.50**±0.50	12.00±0.48
	SQuAD	0.00±0.00	62.11±3.47	68.13±3.58	68.21±3.59	66.28±3.46	63.91±3.32	62.61±3.21	8.61±0.34	9.56±0.38	9.42±0.38	**8.98**±0.36	9.15±0.37
LLaMA3-8B	GSM8k	0.00±0.00	33.11±2.25	53.07±2.68	53.83±2.71	47.79±2.36	51.31±2.54	49.75±2.43	5.61±0.22	4.15±0.17	6.09±0.24	**6.22**±0.25	4.05±0.16
	Spider	0.00±0.00	10.67±1.03	40.04±1.94	41.73±2.08	38.27±1.89	38.14±1.87	36.63±1.75	0.00±0.00	0.57±0.02	1.40±0.06	**1.08**±0.04	0.22±0.01
	PubMedQA	0.00±0.00	29.00±2.12	77.00±3.85	77.00±3.85	72.50±3.54	72.50±3.54	68.00±3.21	9.50±0.38	10.00±0.40	12.50±0.50	**11.00**±0.44	10.50±0.42
	SQuAD	0.00±0.00	73.02±3.94	75.91±4.02	75.20±3.98	67.92±3.61	73.81±3.87	69.12±3.42	11.94±0.48	9.64±0.39	10.48±0.42	**10.01**±0.40	9.71±0.39
Relative Mean Attack Accuracy		-	-	9.58×	9.59×	8.48×	8.43×	8.09×	1.07×	1.09×	1.18×	1.17×	1.00×

4.2 Security Evaluation

Security against Unauthorized Usage. In this subsection, we assess CoreGuard's security against unauthorized usage. We first evaluate its ability to prevent direct unauthorized inference. Then, we assess its security against MS attacks. Table 2 reports the results: in all cases, the unauthorized inference ("Unau") accuracy is 0%, demonstrating CoreGuard's strong resistance to unauthorized inference. In terms of MS attacks, the security of CoreGuard is comparable to the *upper bound*, indicating that attackers cannot misuse the foundational capabilities of the protected model for downstream tasks. Specifically, CoreGuard's relative accuracy is $1.17\times$ compared to the black-box baseline, benefiting from the effective protection provided by our proposed permutation protocal. specifically, the relative accuracy of CoreGuard ($1.17\times$) is similar to that of DTE ($1.18\times$), which protects the same parameters directly using TEE. This suggests that permutation offers a similar security to the strongest protection (i.e., direct protection by TEE). Regarding other methods, the TPTE solution, NPLO ($9.59\times$) offers no defense (no-shield is $9.58\times$), and PPTE solutions, e.g., DarkneTZ ($8.43\times$), only provide weak protection. In contrast, PSP methods, TLG ($1.07\times$) and ShadowNet ($1.09\times$), also reach the upper bound.

Security under Other Attack Settings.

In this subsection, we evaluate CoreGuard's security under various attack settings. Specifically, first, prior evaluations focus on FFT for training. To test CoreGuard under different MS training methods, we assess its security against MS using LoRA, a most commonly used LLM fine-tuning approach. Second, previous evaluations take base models as the deployed model, but in real-world scenarios, LLMs may also be task-customized, which may align with or differ from the attacker's target, which could affect the defense. Third, while CoreGuard approaches the upper bound with the entire dataset, attackers may sometimes only have a small portion of the data. To verify whether it can still approach the upper bound in such cases, where the upper bound is also lower, we assess the defense with varying training data, ranging from 1% to 100%. As shown in Figure 2, we report the attack accuracies under various settings on Qwen2-0.5B, and CoreGuard consistently ensures the model's security across all these settings. The attack accuracies stay close to black-box protection and significantly lower than no-shield cases, regardless of whether the task is aligned, the proportion of data the attacker possesses, or the training method used.

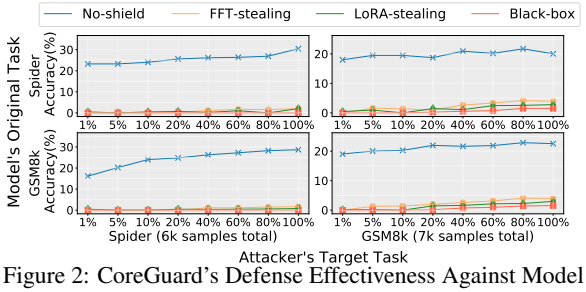

Figure 2: CoreGuard's Defense Effectiveness Against Model Stealing Across Various Attack Settings.

Answer to RQ1: CoreGuard effectively prevents edge-deployed LLMs from being off-device misused, blocking model stealing attacks and achieving upper-bound security across diverse attack settings.

Table 3: The results of additional overhead. For *execution overhead*, we present the original model's FLOPs ("Original"), the additional overhead in TEE ($FLOPs$), and its proportion to the original model's FLOPs ($\%FLOPs$). For *transfer overhead*, we report the transfer volume (KB) and the number of transfers ($rounds$) between the TEE and GPU for each method.

Models	**TEE Execution Overhead** $FLOPs/(\%FLOPs)\downarrow$								**TEE-GPU Transfer Overhead** $KB/rounds\downarrow$						
	Original	Serdab	DarkneTZ	SOTER	**TLG**	**ShadowNet**	DTE	**Ours**	Serdab	DarkneTZ	SOTER	**TLG**	**ShadowNet**	DTE	**Ours**
Qwen2	1.27E+11	3.82E+09 (3.02%)	3.82E+09 (3.02%)	2.59E+10 (20.48%)	**3.54E+07 (2.79e-02%)**	**3.67E+10 (28.97%)**	4.58E+10 (36.23%)	**1.47E+06 (1.17e-03%)**	2.24E+02 2	1.12E+02 2	5.50E+04 67	**1.42E+05 115**	**2.75E+05 336**	2.24E+02 2	**6.18E+03 5**
Gemma2	6.70E+11	1.99E+10 (2.98%)	1.99E+10 (2.98%)	1.44E+11 (21.52%)	**7.67E+07 (1.14e-02%)**	**1.89E+11 (28.23%)**	2.59E+11 (38.72%)	**2.95E+06 (4.41e-04%)**	5.76E+02 2	2.88E+02 2	1.30E+05 72	**3.95E+05 125**	**6.49E+05 364**	5.76E+02 2	**1.58E+04 5**
ChatGLM3	1.53E+12	5.22E+10 (3.41%)	5.22E+10 (3.41%)	3.01E+11 (19.65%)	**2.26E+08 (1.48e-02%)**	**4.86E+11 (31.75%)**	7.31E+11 (47.77%)	**8.06E+06 (5.27e-04%)**	1.02E+03 2	5.12E+02 2	1.78E+05 67	**5.91E+05 135**	**1.02E+06 336**	1.02E+03 2	**2.19E+04 5**
LLaMA3	1.92E+12	5.58E+10 (2.91%)	5.58E+10 (2.91%)	3.85E+11 (20.02%)	**1.51E+08 (7.86e-03%)**	**5.03E+11 (26.15%)**	8.94E+11 (46.50%)	**4.72E+06 (2.46e-04%)**	1.02E+03 2	5.12E+02 2	2.82E+05 90	**6.36E+05 155**	**1.31E+06 448**	1.02E+03 2	**2.05E+04 5**

Table 4: The accuracy comparison between the original model (M_{ori}) and the CoreGuard-protected model (M_{loc}). The result is presented as M_{ori}/M_{loc}. Cells showing changes in accuracy are highlighted in **bold**.

	GSM8k	Spider	PubMedQA	SQuAD
Qwen2	**15.51%/15.50%**	5.56%/5.56%	15.50%/15.50%	16.50%/16.50%
Gemma2	30.10%/30.10%	3.51%/3.51%	10.50%/10.50%	43.21%/43.21%
ChatGLM3	37.13%/37.13%	5.15%/5.15%	46.00%/46.00%	**62.11%/62.09%**
LLaMA3	**33.11%/33.13%**	10.67%/10.67%	**29.00%/28.50%**	**73.02%/73.01%**

4.3 Execution and Transfer Overhead

To answer **RQ2**, we measure both TEE execution and TEE-GPU transfer overheads to assess CoreGuard's efficiency in computation and communication. Specifically, we take an example length of 128 as input and report the TEE execution and data transfer overhead of each solution to generate a single token, excluding TPTE, which offers no protection. All results are reported in Table 3, and CoreGuard demonstrates a clear advantage. Specifically, compared to PPTE solutions, CoreGuard achieves thousands of times lower TEE execution overheads. Specifically, the CoreGuard's execution overhead is less than 1.17e-03% in all cases, whereas PPTE incur execution overheads ranging from 2.91% to 21.52%. More importantly, compared to existing PSP solutions (highlighted in **bold**), CoreGuard's transfer overhead is nearly two orders of magnitude lower due to its communication-friendly design. Specifically, as mentioned in Section 3.3, CoreGuard requires only a single authorization, which is optimal, limiting the transfer rounds to 5. In this way, CoreGuard cuts the unacceptable overhead (seconds per token) of existing PSP solutions by two orders of magnitude to negligible levels.

Answer to RQ2: CoreGuard's computation and communication overheads are significantly lower than other solutions, showcasing a substantial efficiency advantage.

4.4 Accuracy Loss

To answer **RQ3**, we compare the accuracy between the unprotected model M_{ori} and the CoreGuard-protected model M_{loc}. The result is shown in Table 4. As demonstrated, the impact of CoreGuard on accuracy is minimal. Specifically, in most cases, there is no difference in accuracy between M_{ori} and M_{loc}. However, for some specific cases, accuracy slightly fluctuates (highlighted in **bold**). For example, with LLaMA3 on PubMedQA, accuracy decreases slightly by 0.5%. However, interestingly, we observe a 0.02% improvement on GSM8k. Therefore, we consider the minor fluctuations caused by precision limitations (e.g., floating-point errors) rather than the defense itself, which is inevitable.

Answer to RQ3: While significantly outperforming existing defenses in terms of both security and efficiency, CoreGuard maintains the model's accuracy without compromise.

5 Limitation and Discussion

Side Channel Attacks. CoreGuard uses TEEs as its security root, making it vulnerable to side-channel attacks [37, 2]. However, various defense methods have emerged in recent years to mitigate the risk of side-channel leaks, and both these software- [23, 18] and hardware-based [10, 44] defenses can be integrated into our approach. For software-based defense, CoreGuard uses TEE only for basic matrix operations (e.g., matrix permutation and addition). For hardware-based defense, CoreGuard does not require modifications to hardware, allowing physical defense measures to be compatible.

TEE in GPUs. Recent work has explored implementing trusted environments directly within GPUs [26]. However, such GPU/NPU TEEs are still in their early stage and mainly target datacenter settings requiring high-end hardware [27], making them impractical for current edge deployment. Orthogonal to these solutions, CoreGuard instead focuses on broadly available edge devices—such as

https://doi.org/10.52202/085713-2327

smartphones and personal computers—where TEEs are typically CPU-based (e.g., ARM TrustZone, Intel SGX).

Real-World Environments. This paper focuses on the core framework, without delving into device-specific implementations. However, this limitation does not affect the general applicability of our approach. CoreGuard is designed to be hardware-agnostic in both implementation and evaluation. In practice, the TEE masks features, performs permutation, and returns the authorized features, relying only on basic TEE functions like matrix operations and data storage, which are universally supported. Additionally, our evaluation is platform-agnostic and suitable across different platforms.

Architecture Compatibility. CoreGuard is compatible with mainstream transformer architectures, including LLaMA variants and models with Mixture-of-Experts (MoE) design. For MoE-based transformers, CoreGuard protects each expert independently using the same method applied to standard FFNs. The gating mechanism, typically a linear projection, also supports permutation.

6 Conclusions

In this paper, we presents CoreGuard, a protection method that uses a permutation strategy to secure edge-deployed models with maximum security. Importantly, to reduce transfer overhead during authorization, CoreGuard proposes a propagation protocol, thus only a single authorization is required to authorize the entire model, which is optimal. Experimental results show that CoreGuard delivers superior security and efficiency without accuracy loss. In conclusion, CoreGuard is an effective solution that provides model owners with the means to safeguard their proprietary LLMs.

Acknowledgements

This work was sponsored by CCF-Huawei Populus Grove Fund. This work was also supported by the Key Project of the National Natural Science Foundation of China under Grant no. 62536007, the Zhejiang Province Science Foundation under Grant no. LD24F020002 and the Zhejiang Province's 2025 "Leading Goose + X" Science and Technology Plan under Grant no. 2025C02034.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, 2018.

[2] AKM Mubashwir Alam and Keke Chen. Making your program oblivious: a comparative study for side-channel-safe confidential computing. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 282–289. IEEE, 2023.

[3] Tiago Alves. Trustzone: Integrated hardware and software security. *Information Quarterly*, 3:18–24, 2004.

[4] Anthropic. Claude: An ai assistant by anthropic. `https://www.anthropic.com/index/introducing-claude`, 2023. Accessed: 2025-05-12.

[5] Apple Inc. Deploying transformers on the apple neural engine. `https://machinelearning.apple.com/research/apple-intelligence-foundation-language-models`, 2025. Accessed: [2025.05.08].

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] Ying Cao, Ruigang Liang, Kai Chen, and Peiwei Hu. Boosting neural networks to decompile optimized binaries. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 508–518, 2022.

[8] Abhishek Chakraborty, Ankit Mondai, and Ankur Srivastava. Hardware-assisted intellectual property protection of deep learning models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

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

[10] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. {HybCache}: Hybrid {Side-Channel-Resilient} caches for trusted execution environments. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 451–468, 2020.

[11] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.

[12] Tarek Elgamal and Klara Nahrstedt. Serdab: An iot framework for partitioning neural networks computation across multiple enclaves. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 519–528. IEEE, 2020.

[13] Hugging Face. Transformers documentation, 2025. Accessed: 2025-05-05.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1937–1954, 2021.

[16] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*, 2019.

[17] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Paul Leignac, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre, and Simon Pontié. Comparison of side-channel leakage on rich and trusted execution environments. In *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*, pages 19–22, 2019.

[19] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084, 2023.

[20] Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, and Jianwei Yin. Translinkguard: Safeguarding transformer models against model stealing in edge deployment. *arXiv preprint arXiv:2404.11121*, 2024.

[21] Gang Liu, Ruotong Xiang, Jing Liu, Rong Pan, and Ziyi Zhang. An invisible and robust watermarking scheme using convolutional neural networks. *Expert Systems with Applications*, 210:118529, 2022.

[22] Chun Shien Lu. Steganography and digital watermarking techniques for protection of intellectual property. *Multimedia Security, Idea Group Publishing, Singapore*, pages 75–157, 2005.

[23] Tengchao Ma, Changqiao Xu, Qingzhao An, Xiaohui Kuang, Lujie Zhong, and Luigi Alfredo Grieco. A proactive defense strategy against sgx side-channel attacks via self-checking drl in the cloud. In *ICC 2022-IEEE International Conference on Communications*, pages 4174–4179. IEEE, 2022.

[24] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

 https://doi.org/10.52202/085713-2327

[25] Tsunato Nakai, Daisuke Suzuki, and Takeshi Fujino. Towards trained model confidentiality and integrity using trusted execution environments. In *Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings*, pages 151–168. Springer, 2021.

[26] NVIDIA. Nvidia confidential computing. `https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/`, 2024. Accessed: [2024.10.11].

[27] NVIDIA. Nvidia h100 tensor core gpu. `https://www.nvidia.com/en-us/data-center/h100/`, 2024. Accessed: [2024.3.18].

[28] OpenAI. Chatgpt: Optimizing language models for dialogue. `https://openai.com/chatgpt`, 2023. Accessed: 2025-05-12.

[29] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4954–4963, 2019.

[30] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. Activethief: Model extraction using active learning and unannotated public data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 865–872, 2020.

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[32] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[33] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[34] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[35] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. {SOTER}: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 723–738, 2022.

[36] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature Medicine*, pages 1–8, 2025.

[37] Petr Socha, Vojtěch Miškovskỳ, and Martin Novotnỳ. A comprehensive survey on the non-invasive passive side-channel analysis. *Sensors*, 22(21):8096, 2022.

[38] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1596–1612. IEEE, 2023.

[39] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[40] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[41] Timothy Prickett Morgan. Counting the cost of training large language models. `https://www.nextplatform.com/2022/12/01/counting-the-cost-of-training-large-language-models/`, 2022. Accessed: [2024.02.24].

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

[43] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[44] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. Pwrleak: Exploiting power reporting interface for side-channel attacks on amd sev. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 46–66. Springer, 2023.

[45] Ruoyu Wu, Taegyu Kim, Dave Jing Tian, Antonio Bianchi, and Dongyan Xu. {DnD}: A {Cross-Architecture} deep neural network decompiler. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2135–2152, 2022.

[46] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

[47] Guangxuan Xiao, Ji Lin, and Song Han. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*, 2023.

[48] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. Luke: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, 2020.

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

[50] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[51] Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. No privacy left outside: On the (in-) security of tee-shielded dnn partition for on-device ml. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 52–52. IEEE Computer Society, 2023.

[52] Tong Zhou, Yukui Luo, Shaolei Ren, and Xiaolin Xu. Nnsplitter: an active defense solution for dnn model via automated weight obfuscation. In *International Conference on Machine Learning*, pages 42614–42624. PMLR, 2023.

 https://doi.org/10.52202/085713-2327

A Appendix / Detailed Model Stealing Attack

We consider finetuning attacks as a key security challenge of existing protection solutions. The finetuning attack is a commonly used attack model that is widely discussed in previous papers and is a well-defined and reproducible attack that is specifically designed for edge-deployed models. This attack typically occurs in two steps. First, the attacker creates a surrogate model and fills it with all the available parameters from the non-secure world. Then, the attacker trains this surrogate model with the datasets they have access to, attempting to apply the surrogate model to their target task.

Specifically, the attack consists of two phases: *foundational capabilities stealing* (P_1) and *task-specific adaptation* (P_2). In P_1, to exploit the foundational capabilities of the locked model (M_{loc}), the attack begins by inferring the architecture of M_{loc} through its exposed parts. Following this, a replica model, M_{rep}, is constructed with the same architecture as M_{loc}. Finally, the attacker transports M_{loc}'s exposed weights to the corresponding parts of M_{rep}. In P_2, the attacker attempts to fine-tune M_{rep} for their tasks. To this end, one potential approach is to train M_{rep} with the training dataset the attacker possesses. Specifically, We assume the attacker has access to the entire training dataset, making this scenario more challenging than previous works[29, 30, 51], which assume attackers have access to only a small portion (e.g., 1%) of the data. In the training, we mainly consider a more comprehensive and effective method, namely full-parameter training (FFT). However, to ensure comprehensiveness, we also consider other training settings, such as LoRA.

B Appendix / Authorization Position

Security under Different Authorization Position. The position of authorization is a hyper-parameter of CoreGuard, and its selection can influence the overall security. To identify the best authorization position, we examine how different authorization positions impact security. Specifically, we place the authorization before various transformer layers and evaluate their security based on model-stealing attacks.

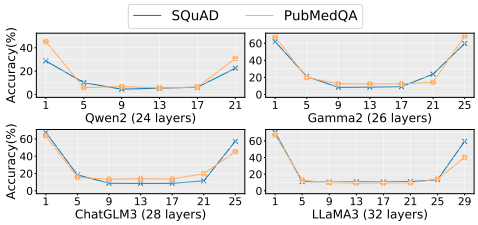

Figure 3: Impact of authorization position on security. Model-stealing accuracy is reported for different positions, with the total number of transformer layers indicated for each model.

As shown in Figure 3, the results demonstrate that placing the initial authorization point in the middle of the network is practical. Specifically, the model stealing accuracy is higher when the authorization is placed near the beginning or end of the network. Conversely, when the authorization is deployed within the central layers of the network, the model-stealing accuracy significantly decreases. This aligns with our expectations: applying authorization at the first layer means CoreGuard permutes nearly all the transformer layers, which means the attacker only needs to recover the functionality of the first transformer layer. In contrast, placing authorization in the middle leaves at least half of the parameters unaligned, requiring the attacker to recover the functionality of at least half of the network's parameters. Thus, placing the authorization in the central layers is an adequate strategy.

C Appendix / Analysis of Neural Network Fitting Attack

The attacker knows that the input and output are $y, m, m', n', \pi z$. Since Linear A is public, it is meaningless to attack the neural network fitting from y to m. In addition, it is meaningless to use m' as input to attack the neural network fitting because m' itself is noisy data. Although m is the same, m' is likely to be different. Therefore, the truly effective input and output of the neural network fitting attack is $m, n', \pi z$.

https://doi.org/10.52202/085713-2327

The attacker's training for neural network fitting attack must be less than the amount of pre-training data, otherwise it will be more trouble than gain. However, the fitted parameters $\hat{W}_b, \hat{\pi}$ are definitely noisy. Therefore, the process can be specifically expressed as

$$m(\hat{\pi}\hat{W}_b) + \epsilon = n' \text{ and Normalization}(m(\hat{\pi}\hat{W}_b)) + \epsilon = \pi z. \tag{13}$$

Define πW_b as a whole W. Then the attacker can crack π and W_b as

$$\hat{\pi}\hat{W}_b + \epsilon = W. \tag{14}$$

In the absence of noise, it is impossible for an attacker to crack it from a mathematical perspective. This is because this is a pathological equation, the number of unknown variables is much greater than the number of equations, and it is a quadratic equation and a nonlinear equation.

From another perspective, the attacker fits infinitely and obtains many $\hat{W}_b$ and $\hat{\pi}$. **Even if we tell the attacker that the $\hat{W}_b$ is closest to the real W_b, it is difficult for the attacker to solve it.** This can be expressed as

$$\hat{\pi}(W_b - \epsilon') + \epsilon = \hat{\pi}W_b + \tilde{\epsilon} = W, \tag{15}$$

where ϵ' is error between W_b and $\hat{W}_b$. Since floating point numbers have a minimum precision, the equation can be converted to integers by dividing both sides by this minimum precision. Such an integer equation solution problem is a Matrix-Learning With Errors problem.

Hardness of Matrix-Learning With Errors (Matrix-LWE). The Matrix-Learning With Errors (Matrix-LWE) problem is a natural generalization of the standard LWE problem. It is defined as follows:

$$W = \hat{\pi}W_b + \epsilon \mod q. \tag{16}$$

If we regard q as the largest positive integer that can be represented by a computer floating point number, then we can directly ignore q. In other words, the attacker's solution to the problem is equivalent to solving the LWE problem.

The goal is to recover the secret $\hat{\pi}$. Matrix-LWE can be viewed as packing ℓ independent instances of the standard LWE problem:

$$W = [W_b\hat{\pi}_1 + \epsilon_1 \mid W_b\hat{\pi}_2 + \epsilon_2 \mid \cdots \mid W_b\hat{\pi}_\ell + \epsilon_\ell] \mod q, \tag{17}$$

where each column corresponds to an individual LWE sample with secret $\hat{\pi}_i$ and noise ϵ_i.

The hardness of Matrix-LWE follows from the hardness of standard LWE. Regev [33] proved that solving LWE on average is at least as hard as solving certain worst-case lattice problems such as the Shortest Independent Vector Problem (SIVP$_\gamma$) and the Gap Shortest Vector Problem (GapSVP$_\gamma$).

Specifically, SIVP$_\gamma$ asks for a set of n linearly independent lattice vectors whose maximum length is at most $\gamma \cdot \lambda_n(\mathcal{L})$, and GapSVP$_\gamma$ asks to decide whether the shortest non-zero vector $\lambda_1(\mathcal{L})$ in a lattice $\mathcal{L}$ is smaller than a given threshold d or larger than γd.

Since these worst-case lattice problems are known to be NP-hard or conjectured to be intractable even for quantum algorithms, Matrix-LWE inherits a strong worst-case to average-case hardness guarantee. In practice, this makes Matrix-LWE a reliable foundation for constructing cryptographic primitives, including post-quantum encryption schemes.

For examply, assume an attacker obtains an approximate estimate $\tilde{W}_b$ of the true parameter matrix W_b satisfying $\|\tilde{W}_b - W_b\|_F \le \varepsilon'$, and observes the authorized (locked) output $W = \pi W_b$. Since π is a permutation matrix in our construction (hence $\|\pi\|_2 = 1$), the attacker can only form

$$W = \pi\tilde{W}_b + \varepsilon, \quad \text{with} \quad \varepsilon := \pi(W_b - \tilde{W}_b),\ \|\varepsilon\|_F \le \varepsilon'.$$

This matches the canonical Matrix-LWE form $W = \pi\tilde{W}_b + \varepsilon$, reducing recovery of π to solving a Matrix-LWE instance.

 https://doi.org/10.52202/085713-2327

D Appendix / Adaptive Attack

Security against Permutation Matrix Simulation Attack. In this subsection, we assess the security of CoreGuard against attackers familiar with the CoreGuard's mechanism and implement attacks accordingly. Specifically, the core of authorization involves the permutation matrix (i.e., π), which the TEE protects. Therefore, an attacker might first attempt to simulate π by initializing a substitute permutation matrix with the same shape and training it based on the TEE's input and output to approximate the true π. Then, the attacker uses the simulated π to mimic the TEE's authorization and fine-tunes the model on their task to complete the attack.

Table 5: Security evaluation of CoreGuard against permutation matrix simulation attack (*simulation*).

		No-Shield	***Simulation***	**Black-box**
Qwen2	GSM8k	21.53%$\pm$1.43%	0.00%$\pm$0.00%	1.29%$\pm$0.03%
	PubMedQA	58.00%$\pm$2.56%	3.50%$\pm$0.52%	5.00%$\pm$0.20%
Gamma2	Spider	39.15%$\pm$1.71%	0.00%$\pm$0.00%	0.00%$\pm$0.00%
	SQuAD	63.96%$\pm$3.08%	0.00%$\pm$0.00%	8.81%$\pm$0.35%
ChatGLM3	GSM8k	55.95%$\pm$2.87%	0.00%$\pm$0.00%	0.23%$\pm$0.01%
	PubMedQA	71.00%$\pm$3.34%	1.00%$\pm$0.76%	12.00%$\pm$0.48%
LLaMA3	Spider	40.04%$\pm$1.94%	0.00%$\pm$0.00%	0.22%$\pm$0.01%
	SQuAD	75.91%$\pm$4.02%	3.18%$\pm$0.61%	9.71%$\pm$0.39%

The results are shown in Table 5; the attack is ineffective, even performing worse than the black-box baseline. The outstanding security is due to the targeted design. Specifically, the non-linear nature of the authorization process, which relies on π, significantly increases the difficulty of the simulation. Moreover, CoreGuard requires high precision in the authorization process, where even slight simulation errors can compromise model performance.

Security against Authorization Simulation Attack. Considering that precisely fitting π is a challenging task, we consider that attackers might attempt to extend their simulation to include adjacent layers or structures, potentially making the attack more feasible. Specifically, since the TEE and the FFN block jointly achieve the authorization, they can be considered as a single unit, which the attacker might attempt to simulate directly. Therefore, the attacker could reconstruct an FFN block structure and train this new FFN block based on the input and output of the original TEE-authorized FFN block, thereby bypassing the TEE's authorization.

Table 6: Security evaluation of CoreGuard against authorization simulation attack (*simulation*).

		No-Shield	***Simulation***	**Black-box**
Qwen2	GSM8k	21.53%$\pm$1.43%	2.72%$\pm$0.61%	1.29%$\pm$0.03%
	PubMedQA	58.00%$\pm$2.56%	7.00%$\pm$1.66%	5.00%$\pm$0.20%
Gamma2	Spider	39.15%$\pm$1.71%	0.00%$\pm$0.00%	0.00%$\pm$0.00%
	SQuAD	63.96%$\pm$3.08%	3.51%$\pm$0.15%	8.81%$\pm$0.35%
ChatGLM3	GSM8k	55.95%$\pm$2.87%	0.00%$\pm$0.00%	0.23%$\pm$0.01%
	PubMedQA	71.00%$\pm$3.34%	13.50%$\pm$0.58%	12.00%$\pm$0.48%
LLaMA3	Spider	40.04% $\pm$1.94%	0.00%$\pm$0.00%	0.22%$\pm$0.01%
	SQuAD	75.91%$\pm$4.02%	6.81%$\pm$0.72%	9.71%$\pm$0.39%

As shown in Table 6, the attack is ineffective. In all cases, the attack accuracy is similar between the simulation and the black-box baseline but significantly lower than the no-shield baseline. The outstanding defense effectiveness is due to the targeted design. Specifically, CoreGuard disrupts the alignment of the parameters before and after the authorization, making it highly challenging for attackers to simply adjust the FFN to recover the compatibility between the two sets of parameters.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction of the paper clearly outline the main contributions and the scope of the research.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We claimed the limitation in Section 5.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

https://doi.org/10.52202/085713-2327

Justification: Every theorem stated in the main text is accompanied by a complete proof provided in the appendix, with appropriate cross-references to ensure clarity and rigor.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experiments can be reproduced and the paper provides a clear and comprehensive explanation of the proposed method.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We will release code as soon as the article is accepted.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We claimed the experimental details in Section 4.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the error margins alongside all experimental results to reflect uncertainty and ensure transparency.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

https://doi.org/10.52202/085713-2327

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the GPU used in our experiments as well as its memory capacity.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This work investigates model merging stealing, an emerging form of model theft that may pose a serious threat to the open-source model ecosystem.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The assets used in this paper are properly credited, and the license and terms are respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

 https://doi.org/10.52202/085713-2327

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: We did not conduct any research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The LLM is only used for writing.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

https://doi.org/10.52202/085713-2327