Game of Arrows: On the (In-)Security of Weight Obfuscation for On-Device TEE-Shielded LLM Partition Algorithms

Pengli Wang[1], Bingyou Dong[2], Yifeng Cai[1], Zheng Zhang[2], Junlin Liu[1], Huanran Xue[2], Ye Wu[2], Yao Zhang[2], and Ziqi Zhang[3,*]

[1]*MOE Key Lab of HCST (PKU), School of Computer Science, Peking University*
[2]*ByteDance*
[3]*University of Illinois Urbana-Champaign*
[*]*Corresponding Author*

Abstract

Utilizing Trusted Execution Environments (TEEs) to protect Large Language Models (LLMs) on users' devices is a practical solution for model owners. To alleviate the computation burden on TEEs, researchers have proposed TEE-Shielded LLM Partition (TSLP) to offload heavy computation layers to co-operating untrusted GPUs, while lightweight layers are shielded in TEE. TSLP utilizes various lightweight obfuscation schemes to protect offloaded weights from various attacks meanwhile not introducing large computation overhead. However, existing lightweight obfuscation algorithms have one vital vulnerability in common: the direction similarity of obfuscated vectors. In this paper, we propose a novel attack, ARROWMATCH, that utilizes direction similarity to recover obfuscated private weights. To achieve this, ARROWMATCH compares direction distances between obfuscated model weights and public pre-trained model weights. To mitigate this vulnerability, we propose a novel obfuscation scheme, ARROWCLOAK, which leverages lightweight matrix-vector multiplication to protect vector directions and private weights. We evaluate ARROWMATCH and ARROWCLOAK on four representative LLMs, using seven datasets, along with five obfuscation schemes. The results show that ARROWMATCH can break the protection of all existing lightweight obfuscation schemes with high accuracy (similar to no protection) and effectively recover the private weights (with over 98% accuracy). In addition, ARROWCLOAK can effectively defend against ARROWMATCH (6.5× better than state of the art) and protect direction information by increasing the direction distance over 900×. We also evaluate the performance of ARROWCLOAK on a real-world Intel SGX device and show that ARROWCLOAK can reduce total overhead by 2.83× compared to shield-the-whole baseline.

1 Introduction

On-Device LLM Security. Large Language Models (LLMs) have become important intellectual properties for AI companies [18, 19]. To provide efficient inference services, tech giants choose to deploy LLMs on users' devices [33]. However, this approach exposes the model to potential attackers who could steal the weights of the deployed models using hardware [46, 85] or software [38] techniques. The on-device deployment method introduces a new attack surface that would allow adversaries to access *white-box* model information compared to cloud-based deployment, where adversaries are supposed to have only *black-box* access [57, 58, 83].

TEE-Shielded LLM Partition (TSLP). Trusted Execution Environments (TEEs) can be utilized to protect the model weights in on-device deployment settings. The goal of TEE-based defense is to downgrade leakage vulnerabilities from a white-box model to a black-box model [57, 58, 83]. However, existing commercial TEE products suffer from limited computational capabilities and capacities. To resolve this problem, researchers have proposed TEE-Shielded LLM partition (TSLP) solutions. TSLP offloads computation-intensive layers to untrusted GPUs (in the host OS) and deploys other layers inside TEE [31, 58, 62]. This strategy, which can avoid performing complex operations in TEEs, helps to reduce the total overhead of LLM inference.

Weight Obfuscation in TSLP. TSLP protects the offloaded weights by adding obfuscations before sending them to unprotected GPUs. Such obfuscation techniques play a critical role in preventing attackers from recovering model weights meant to be private. The design of obfuscation algorithms aims to find a balance between security (for the obfuscated weights) and efficiency (for output reconstruction in the TEE). We did a comprehensive summary of nine obfuscation algorithms for TSLP and have found that existing works mainly use *per-vector* [1] operations, namely vector permutation [31, 76] and vector scaling [52], which are applied in TEE to obfuscate the weight matrix. The only exception (GroupCover [82]) uses matrix multiplication to linearly combine weight vectors, which introduces heavier computation overhead than other lightweight schemes according complexity analysis.

[1] A vector means a column in the weight matrix.

Per-vector operations prioritize efficiency and apply the same operation evenly to all values in each vector. However, the limitation of per-vector operations is that they can not protect the vector directions. Vector permutations would only change the position of each vector in the weight matrix, while vector scaling only changes the vector length. These operations have no impact on directions, leading to the fact that the directions of vectors can be recovered by attackers even after obfuscation, which indicates potential vulnerability.

Insight: Direction Similarity. We find that adversaries can utilize the direction similarity between an obfuscated model (M_{obf}) and a public pre-trained model (M_{pre}) to break the lightweight obfuscation algorithms. This similarity can be used to find the vector matches (correct pairs) between M_{obf}'s weight vectors and M_{pre}'s weight vectors. The direction distances of correct matches are significantly lower than those of incorrect weight matches. Thus, by comparing the direction distances of different weight vector pairs, we can recover the correct matches and further recover the private weights. Our insight is based on two observations: low direction discrepancy during victim model training (**Obs1**) and direction invariance of obfuscation algorithms (**Obs2**). **Obs1** is because private models need to reuse foundation models' general knowledge to achieve high performance [16] and retain the vector directions [3,24]. **Obs2** is because existing obfuscation algorithms are per-vector based and do not change vector directions due to efficiency considerations.

Attack: ARROWMATCH. Based on our insight, we propose a novel attack called ARROWMATCH. The goal of ARROWMATCH is to construct a surrogate model (M_{sur}) which has similar functionalities as the victim model (M_{vic}). ARROWMATCH consists of two stages: distance-based direction recovery (**S1**) and learning-based vector length adjustment (**S2**). In **S1**, we recover the vector permutation by selecting the vector with the smallest direction distance to the public weight vectors. In **S2**, we adjust the vector length to match the public weights and train the surrogate model with a small amount of data to achieve similar functionality as M_{vic}.

Defense: ARROWCLOAK. To mitigate the attack, we propose a novel lightweight obfuscation algorithm, ARROWCLOAK, which aims to protect vector directions of private weights. The insight of ARROWCLOAK is to use *matrix-vector multiplication* in the TEE to obfuscate the weight matrix. Matrix-vector multiplication is as lightweight as per-vector operations, the computation complexity of which is over two magnitudes lower than matrix-matrix multiplication. This operation can effectively randomize vector directions. ARROWCLOAK consists of two stages. The first stage obfuscates the weight matrix by multiplying the matrix with a random vector in the TEE. The second stage uses the obfuscated weights to infer each LLM layer in a heterogeneous manner. We also demonstrate that our ARROWCLOAK can be reduced to the LWE problem [48], a well-known hard problem in cryptography. We analyze the numerical value distribution of 'security parameters' to demonstrate that it satisfies the requirement of the LWE problem.

Evaluation We evaluate ARROWMATCH and ARROWCLOAK on four representative LLM models using seven datasets. The models include BERT, GPT-2, and ViT. These datasets come from both computer vision and text tasks. We follow prior work and implement five state-of-the-art obfuscation algorithms. Evaluation results show that ARROWMATCH breaks the defense effect of all prior lightweight obfuscation algorithms and recovers the model weights with high accuracy. The performance of ARROWMATCH is 1.67× better than the black-box attack, which is quite close to that of a no-shield white-box attack (1.70×). The accuracy of **S1** and **S2** are above 98%. We also demonstrate the effectiveness of ARROWCLOAK. Under ARROWCLOAK setting, the performance of ARROWMATCH is a close match to Shield-Whole (only 10% higher privacy leakage) setting and is 6.5× lower than the best performance of other obfuscation algorithms. ARROWCLOAK can also protect the direction of private weight vectors effectively. The direction distance between the vectors before and after obfuscation is 961.94× higher than prior work. We also evaluate utility experiments on a real-world machine with Intel SGX, which is one of the most widely used TEEs. We implement ARROWCLOAK on a public framework and evaluate the inference latency. The results show that ARROWCLOAK can reduce the inference latency by 2.83× compared to Shield-Whole. The obfuscation overhead of ARROWCLOAK is only 0.46× compared to a non-obfuscated TSLP solution, which demonstrates the efficiency of our solution.

We summarize the contributions as follows :

- We propose a novel attack, ARROWMATCH, which utilizes direction similarity to break the defense of existing lightweight obfuscation algorithms.
- We propose a novel obfuscation algorithm, ARROWCLOAK, to protect vector directions in the private weight matrix.
- We evaluate ARROWMATCH and ARROWCLOAK on four representative LLMs, seven datasets, with five obfuscation schemes. The results show that ARROWMATCH can break the defense of all prior lightweight obfuscation algorithms and ARROWCLOAK can effectively prevent the attack.

2 Background

In this section, we will introduce the background of this paper, including Large Language Model (LLM; Sec. 2.1), Model Stealing (MS; Sec. 2.2), TEE-Shielded LLM Partition (TSLP; Sec. 2.3) and weight obfuscation for TSLP(Sec. 2.4)

2.1 Large Language Model (LLM)

Transformer. The most important LLM architecture is the transformer architecture [63]. Each LLM is constructed by stacking several transformer blocks. Each block consists of two modules: an attention module and a feed-forward (FF) module.

Operation Analysis. We extend prior work [58, 62, 83] to categorize the operations into three operation types based on the complexity w.r.t the input vector. The categories are linear operations ($\texttt{Op}_{\texttt{linear}}$), quadratic operations ($\texttt{Op}_{\texttt{quadra}}$), and non-polynomial operations ($\texttt{Op}_{\texttt{non-poly}}$). $\texttt{Op}_{\texttt{linear}}$ refers to the matrix multiplication between model weights and input matrix. $\texttt{Op}_{\texttt{quadra}}$ refers to the matrix multiplication between attention matrices. $\texttt{Op}_{\texttt{non-poly}}$ includes other operations, such as the activation function. We use n to represent the dimension of the internal feature and l to represent the sequence length. The input and output are $X, Y \in \mathbb{R}^{n \times l}$. The weight matrix for $\texttt{Op}_{\texttt{linear}}$ is $W \in \mathbb{R}^{n \times n}$. The computation complexity of $\texttt{Op}_{\texttt{linear}}$ is $O(n^2 l)$. $\texttt{Op}_{\texttt{non-poly}}$ only takes marginal computation (less than 8%). $\texttt{Op}_{\texttt{linear}}$ and $\texttt{Op}_{\texttt{quadra}}$ takes about 66% and 25% computation operations, respectively [60, 62].

2.2 Model Stealing

Model Stealing (MS) attack is a real-world threat to the on-device LLM models. For a victim private model M_{vic}, the goal of MS is to construct a surrogate model M_{sur} that has a similar functionality as M_{vic}. MS attacks can be categorized into two types: white-box attacks and black-box attacks. Black-box attacks collect a large amount of training data from the M_{vic} and directly train M_{sur} [41]. White-box attacks utilize system or hardware access to directly steal M_{vic}'s weights and construct M_{sur} [58, 83]. Some white-box attack surfaces include memory [26, 46] and PCIe bus [85]. The attacker first steals partial M_{vic} weights and constructs M_{init}. Then the attacker collects a small amount of labeled data from M_{vic} and trains M_{init} to get M_{sur}. *White-box attacks are cheaper, more effective, and more efficient than black-box attacks because the adversary can directly steal M_{vic}'s weights* [83].

2.3 TEE-Shielded LLM Partition (TSLP)

TEE-based Model Protection. We follow prior work to regard TEE as *a secure and computation-limited area (without GPU) on a malicious adversary host machine* [23, 36, 53, 58, 82, 83]. Although there are some prototypes of GPU TEE in both academic community [55, 68] and industrial products (e.g. Nvidia's Hopper [40]), they are not widely commercially available [71]. This paper focuses on existing commercial TEEs without GPU because we want to provide a practical solution for real-world deployment. TEE can be utilized to protect LLM models on users' devices [58, 83]. By shielding the model inside TEE, the host machine can not see the plaintext of the model weights. The goal of TEE-based protection is to *downgrade the efficient and cheap white-box attacks against slower and more expensive black-box attacks* [23, 36, 52, 58, 82, 83]. This way, the data owner can reduce the additional attack surface (white-box access) exposed to the host machine.

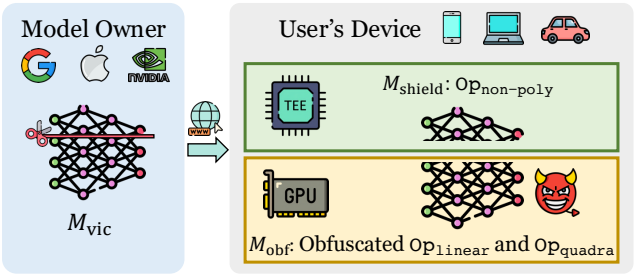

Figure 1: An illustration of TSLP.

TSLP. TSLP aims to mitigate the computation limitation of TEE [23, 36, 53, 58, 82, 83]. Because directly shielding LLM will introduce non-trivial overhead (sometimes up to 50× [62]), TSLP utilizes the co-located *untrusted GPU* to offload part of the LLM computation. TSLP partitions the model into two parts. One is a computation-light part and is shielded by TEE to provide security protection (M_{shield}). The other is an obfuscated computation-intensive part and offloaded to the untrusted GPU for fast computation (M_{obf}). Typically, TSLP solutions select $\texttt{Op}_{\texttt{non-poly}}$ as the M_{shield} and leave $\texttt{Op}_{\texttt{linear}}$ and $\texttt{Op}_{\texttt{quadra}}$ in M_{obf}. Fig. 1 illustrates the TSLP design and the partition process.

Integrity Check and Intermediate Result Protection. Prior work has proposed solutions to check the integrity of untrusted GPU's computation on M_{obf} and protect the input/output sent outside of TEE [58, 62, 83]. In this paper, we focus on the weight obfuscation problem and reuse the integrity check and intermediate result protection from prior work. We leave a more detailed introduction of these solutions in Appx. A.

2.4 Weight Obfuscation for TSLP

TSLP solutions use various weight obfuscation techniques to protect the weight values of M_{obf} [52, 57, 58]. We first formulate the weight obfuscation algorithms and summarize existing literature.

Formulation. We focus on $\texttt{Op}_{\texttt{linear}}$ because only $\texttt{Op}_{\texttt{linear}}$ directly relates to model weights. For each weight matrix W, we take a column view and split W into multiple columns. We regard each column as a vector: $W = [\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_n]$, abbreviated as $W = [\mathbf{w}_i]$ (n is the matrix dimension). Let $Y_{\text{vic}} = f(X; W_{\text{vic}}) = X \cdot W_{\text{vic}}$ represent $\texttt{Op}_{\texttt{linear}}$, where $W_{\text{vic}} \in \mathbb{R}^{n \times n}$ is the private model weights. The defender uses an obfuscation function $g(\cdot) : \mathbb{R}^{n \times n} \Rightarrow \mathbb{R}^{n \times n}$ to compute an obfuscated weight matrix $W_{\text{obf}} = g(W_{\text{vic}})$. TEE sends W_{obf} and X[2] to untrusted GPU. GPU computes the results over W_{obf}

[2]As illustrated in Sec. 2.3, X is protected before sent outside of TEE. For

Table 1: We compare existing weight obfuscation schemes in TSLP. We analyze the obfuscation and de-obfuscation functions, the annotation of the obfuscation keys, the complexity of de-obfuscation, and the TEE workload. The capability of vector direction protection is analyzed in Sec. 4.1. ARROWCLOAK is our proposed defense in Sec. 6.

Paper	Venue	Obfuscation and De-obfuscation Function	Annotation	Obfuscation Keys	Deobfuscation Complexity	Small TEE Workload	Protect Vector Direction
SOTER [52]	ATC'22	$g(W_{\text{vic}}) = \mu \cdot W_{\text{vic}}$, $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}}/\mu$	μ is a periodically updated scalar	μ	$O(nl)$	✓	✗
ShadowNet [58]	S&P'23	$g(W_{\text{vic}}) = \Lambda \cdot W_{\text{vic}} + F$, $g^{-1}(Y_{\text{obf}}) = \Lambda^{-1} \cdot (Y_{\text{obf}} - xF)$	$\Lambda = D \cdot \Pi$, D is a diagonal matrix, Π is a permutation matrix, $x \cdot F$ is computed in $f(x; W_{\text{obf}})$	D, Π, F	$O(nl)$	✓	✗
GroupCover [82]	ICML'24	$g(W_{\text{vic}}) = [A_1 W^{(1)}, \cdots, A_k W^{(k)}] \cdot \Pi$, $g^{-1}(Y_{\text{obf}}) = ([A_1^{-1} Y_{\text{obf}}^{(1)}, \cdots, A_k^{-1} Y_{\text{obf}}^{(k)}]) \cdot \Pi^{-1}$	$A_i \in \mathbb{R}^{\frac{n}{k} \times n}$, k is the number of groups, $W^{(i)}$ represents n/k vectors by clustering W_{vic}	A_i, Π	$O(n^2 l/k)$	✗	✓
TransLinkGuard [31]	MM'24	$g(W_{\text{vic}}) = W_{\text{vic}} \cdot \Pi$, $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}} \cdot \Pi^{-1}$	π is specific for each layer	Π	$O(nl)$	✓	✗
Tempo [76]	IJCNN'24	$g(W_{\text{vic}}) = \vec{c} \cdot W_{\text{vic}} \cdot \Pi$	$\vec{c} \in \mathbb{R}^n$, π is a permutation matrix	$\vec{c}, \Pi$	$O(nl)$	✓	✗
KV-Shield [78]	MobiArch'24	$g(W_{\text{vic}}) = W_{\text{vic}} \cdot \Pi$, $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}} \cdot \Pi^{-1}$	Π is a permutation matrix	Π	$O(nl)$	✓	✗
TSQP [57]	S&P'25	$g(W_{\text{vic}}) = W_{\text{dissim}}/s$, $g^{-1}(Y_{\text{obf}}) = s \cdot Y_{\text{obf}}$,	W_{dissim} is a learned weight to optimize dissimilarity, s is quantization scalar	s, $W_{\text{vic}} - W_{\text{dissim}}$	$O(nl)$	✓	✗
CoreGuard [32]	Arxiv'24	$g(W_{\text{vic}}) = W_{\text{vic}} \cdot \Pi$, $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}} \cdot \Pi^{-1}$	Π is shared by all layers	Π	$O(nl)$	✓	✗
ARROWCLOAK	-	$g(W_{\text{vic}}) = (W_{\text{vic}} \cdot D_1 + \mathbf{v} \cdot \mathbf{1}_n \cdot D_2) \cdot \Pi$ $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}} \cdot \Pi^{-1} \cdot D_1^{-1} - X \cdot \mathbf{v} \cdot \mathbf{1}_n \cdot D_2 \cdot D_1^{-1}$.	D_1 and D_2 are diagonal matrices, $\vec{v} = \sum k_j \cdot \vec{w}_j$, Π is a permutation matrix	$D_1, D_2, \vec{v}, \Pi$	$O(nl)$	✓	✓

by performing $Y_{\text{obf}} = f(X; W_{\text{obf}})$ and send Y_{obf} to TEE. TEE uses a deobfuscation function $g^{-1}(\cdot)$ to recover the results $Y_{\text{vic}} = g^{-1}(Y_{\text{obf}})$.

$g(\cdot)$ should be designed to have low computation complexity and low TEE workload: $O(g(W_{\text{vic}}) + g^{-1}(Y_{\text{obf}})) \ll O(f(X; W_{\text{vic}}))$. This is because TSLP aims to avoid heavy matrix multiplication inside TEE. The computation complexity of $f(X; W_{\text{vic}})$ is $O(n^2 l)$, thus the complexity of the obfuscation algorithm should be much lower than $O(n^2 l)$. Otherwise, if $O(g(W_{\text{vic}}) + g^{-1}(Y_{\text{obf}})) = O(f(X; W_{\text{vic}}))$, the defender can directly perform $f(X; W_{\text{vic}})$ in the TEE and does not need to offload.

Literature Summary. We summarize the weight obfuscation literature and algorithms in Tab. 1. Tab. 1 displays the obfuscation function $g(\cdot)$ and de-obfuscation function $g^{-1}(\cdot)$, and the workload for each algorithm. We can observe that, except for GroupCover, all the other algorithms have a low computation complexity ($O(nl)$) and low TEE workload. We call the algorithms with low computation complexity ($O(nl)$) lightweight obfuscation algorithms. Because LLMs usually have a large n (e.g., 768 or 1024 [13]), the complexity of $O(nl)$ is at least two magnitudes lower than $O(n^2 l)$. GroupCover has a high complexity of $O(n^2 l/k)$, where k is a small constant. Thus GroupCover has a high TEE workload.

Common Characteristics. The common motivation of these lightweight algorithms is that *they use per-vector operations and avoid matrix multiplication to achieve $O(nl)$ complexity*. From Tab. 1, we can observe that lightweight obfuscation algorithms mainly rely on two types of matrices: the permutation matrix Π and diagonal matrix D. Permutation matrix Π changes the relative vector positions in W_{vic}. Let $\pi(\cdot)$ represent the permutation function for Π and maps i-th vector to $\pi(i)$. For each $\mathbf{w}_{\text{vic}}^i \in W_{\text{vic}}$, Π works as $W_{\text{vic}} \cdot \Pi = [\mathbf{w}_{\text{vic}}^i] \cdot \Pi = [\mathbf{w}_{\text{vic}}^{\pi(i)}]$. The diagonal matrix D scales all vectors in W_{vic} with different scalars. Other scaling techniques scaling (used by SOTER [52] and Tempo [76]) can be generalized to D. Although some algorithms use different operations, they can be easily removed or converted into similar operations mentioned before.

simplicity, we do not explicitly mention this protection in the main paper.

3 Threat Model

In this section, we will introduce the threat model of this paper. The threat model follows previous TSLP work that uses TEE to protect model security on users' devices [58, 82, 83].

Scenario. We study a two-party LLM inference scenario. One is the model owner (AI service providers), and the other is the model user (*e.g.* financial institutions). The model provider deploys the model to users' devices. Some examples include smartphone applications [33] and autonomous driving systems [85]. We assume the users are *honest-but-curious adversaries* who are willing to follow the deployment protocol but want to steal the deployed model [57, 58]. The model owner is the defender to protect model security on users' devices. Following [29, 57, 82, 83], we assume victim models are fine-tuned from public LLMs due to prohibitive scratch training costs and the imperative need for task-specific customization (e.g., alignment [22])

Adversary's Goal. As discussed in Sec. 2.2, the adversary's goal is to steal the functionality of the deployed model. After stealing the model functionality, the adversary can directly use it without paying fees or selling the model to others. Specifically, the adversary aims to construct a surrogate model M_{sur} that has the same functionality as the victim model M_{vic} [58].

Adversary's Capability. We assume the adversary can control the user device and access all model data outside of TEE. Some access methods include memory scanning [46], installa-

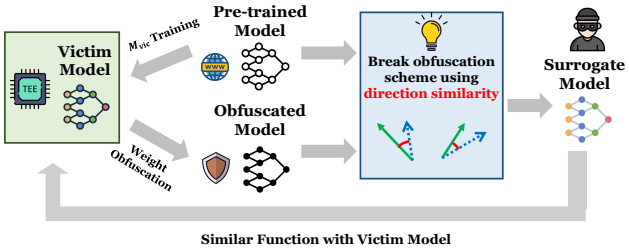

Figure 2: The adversary uses direction similarity between M_{pre} and M_{obf} to construct M_{sur}, which can steal the functionality of M_{vic}.

tion package unpacking [59], and bus monitoring [85]. However, the adversary has no access to the code or data in the TEE. We assume the adversary can use public pre-trained models M_{pre} and data to help analyze the obfuscated model weights. This is a common and practical assumption in model stealing attacks [41,82]. The adversary can use existing fingerprint attacks to identify the pre-trained model and download it from the Internet [11, 66]. We also assume the adversary can query M_{vic} to collect a small amount of labeled data (less than 1%) to train M_{sur} [26, 46, 77].

Defender's Goal and Capability. The defender aims to *downgrade the white-box attacks to black-box attacks*. We consider the black-box defense as the security upper-bound [82, 83]. TSLP solutions do not aim to completely defend against black-box attacks because black-box attacks are still possible even if the model is deployed to the owner's server. TSLP aims to eliminate additional security leakage during model deployment. The model owner can control the data and code in TEE to protect the model's security. The defender can design the weight obfuscation algorithms to protect M_{obf}.

4 Direction Similarity

In this section, we introduce the central insight of this paper: the adversary can utilize the direction similarity to break the defense of weight obfuscation schemes. We first illustrate this insight in Sec. 4.1. The insight is based on two observations. In Sec. 4.2, we will illustrate the two observations.

4.1 Attack Insight

Insight. Our attack insight is that there exists direction similarity between the weight vectors in the obfuscated weights W_{obf} and the publicly available pre-trained weights W_{pre}. The adversary can utilize the direction similarity to recover the permutation and scale operations, and break the obfuscation schemes. The adversary can recover weight information from the obfuscated model and construct the surrogate model M_{sur}. The surrogate model has a similar functionality as the TEE-shielded victim model M_{vic}, thus stealing the functionality of M_{vic}. This insight is based on two observations. The first observation is the low direction discrepancy between the weight vectors of W_{vic} and W_{pre} (**Obs1**). The second observation is the direction invariance of the weight vectors in W_{obf} (**Obs2**).

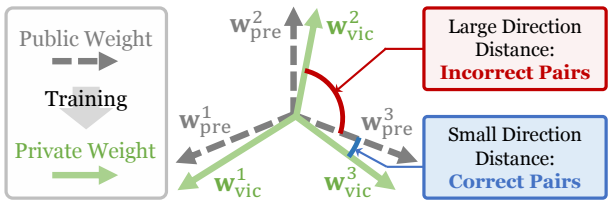

Figure 3: The low direction discrepancy in M_{vic} training.

Fig. 2 shows the insight of this paper. The model owner first trains M_{vic} from M_{pre}. During this phase, the weights of M_{vic} can not deviate too much from M_{pre} (**Obs1**) because M_{vic} needs to reused the capability inside M_{pre}. After M_{vic} is deployed in TEE, the model owner uses a weight obfuscation algorithm to protect the model. We found that existing lightweight obfuscation algorithms do not change the vector direction in the weight matrix (**Obs2**). This is because these algorithms need to guarantee high computation efficiency of obfuscation and de-obfuscation in TEE. Changing the vector direction requires performing complex matrix multiplication and introducing non-trivial overhead. Thus, the weight vector directions between M_{pre} and M_{obf} are similar. The adversary can utilize this similarity to break the obfuscation scheme and construct M_{sur}.

Symbol Definition. We will use W_{vic}, W_{obf}, and W_{pre} to represent the weight matrix of the victim model M_{vic}, obfuscated model M_{obf}, and pre-trained model M_{pre}, respectively. We use $\mathbf{w}^i_{\text{vic}} \in W_{\text{vic}}$, $\mathbf{w}^i_{\text{obf}} \in W_{\text{obf}}$, and $\mathbf{w}^i_{\text{pre}} \in W_{\text{pre}}$ to represent the i-th vector in W_{vic}, W_{obf}, and W_{pre}, respectively.

4.2 Two Observations

In this section, we illustrate why the two observations exist and are not easy to eliminate.

Obs1: Low Direction Discrepancy. We illustrate the first observation in Fig. 3. In the figure, we use three arrows to represent the weight vectors of each model. The gray dashed arrows represent M_{pre}, and the green solid arrows represent M_{vic}. After M_{vic} training, the direction of M_{vic}'s arrows are only slightly different from M_{pre}. For example, $\mathbf{w}^3_{\text{vic}}$ and $\mathbf{w}^3_{\text{pre}}$ have similar directions. We call the vector pair $(\mathbf{w}_{\text{pre}}, \mathbf{w}_{\text{vic}})$ where $\mathbf{w}_{\text{vic}}$ is trained from $\mathbf{w}_{\text{pre}}$ as the correct pair. Otherwise, it is an incorrect pair. As shown in the figure, $(\mathbf{w}^3_{\text{pre}}, \mathbf{w}^3_{\text{vic}})$ is a correct pair, and $(\mathbf{w}^3_{\text{pre}}, \mathbf{w}^2_{\text{vic}})$ is an incorrect pair. The direction distance within a correct pair is smaller than that within an incorrect pair.

This is because M_{vic} relies on M_{pre}'s general capability to perform downstream tasks. M_{vic} can not deviate from M_{pre} too much. Thus, the training process also needs to minimize

the change and maintain W_{pre} as much as possible [16]. Prior work has demonstrated that the weight updates during LLM training exhibit the "low intrinsic dimension" property and can be projected to a smaller subspace [3]. The following work further demonstrated that the weight updates can be decomposed into low-rank spaces (the rank can be as low as 0.5% of the original matrix) [24], which means that the modification to the vector direction is limited. Some recent work also uses theoretical analysis to demonstrate that the weight difference before and after training is bounded by the norm of decomposed matrics [16], which means the direction change is also bounded.

Observation Validation. We conduct a preliminary experiment to demonstrate **Obs1**. We use three LLMs and four datasets (details in Sec. 7.1). We use the distance over unit vectors to represent the direction distance. To guarantee the generalization, we select three different distance functions: cosine distance, L_2 distance, and L_∞ distance. For each $\mathbf{w}^i_{\text{vic}}$, we measure the direction distance with three types of vectors: `Original`, `Second-Cloest`, and `Random`. `Original` is the original vector $\mathbf{w}^i_{\text{pre}}$. `Second-Cloest` is the W_{pre}'s vector that has the second smallest distance to $\mathbf{w}^i_{\text{vic}}$ (the vector with the smallest distance should be $\mathbf{w}^i_{\text{pre}}$). `Random` is a randomly selected vector (other than $\mathbf{w}^i_{\text{pre}}$) in W_{pre}. We sample a random vector 100 times and compute the average distance. The results are shown in Tab. 2. We can observe that the distance of `Original` is much smaller than those of `Second-Closest` and `Random`. For example, the cos of `Original` is 0.0006, but the distances of `Second-Closest` and `Random` are larger than 0.79. L_2 and L_∞ also show the same trend. Averagely, the distance of `Second-Closest` is 77.82× larger than `Original`, and the distance of `Random` is 92.54× larger.

Table 2: Validation for **Obs1**. We report results of three distance functions.

Distance	cos	L_2	L_{inf}	Average
`Original`	0.0006	0.0238	0.0033	0.0093(1.00×)
`Second-Closest`	0.7999	1.2616	0.1095	0.7237(77.82×)
`Random`	0.9951	1.4093	0.1775	0.8606(92.54×)

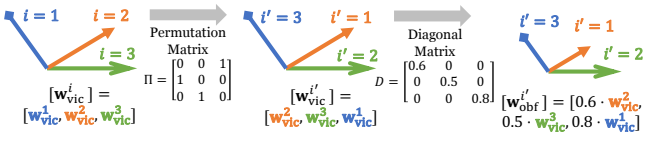

Figure 4: Direction invariance of per-vector operations. Best viewed in color. We use three arrows (blue, orange, and green) to represent three weight vectors. For each vector, i is the index in W_{vic}, and i' is the index in W_{obf}. After applying Π and D to W_{vic}, the vector direction remains the same.

Obs2: Direction Invariance. The limitation of lightweight obfuscation algorithms is that *they can not obfuscate the vector direction.* Fig. 4 shows the effect of per-vector operations on three vectors. Π changes the vector positions. For example, $\mathbf{w}^1_{\text{vic}}$ (the blue arrow) becomes the third vector (in the middle figure). Thus its new index i' is 3. $\mathbf{w}^2_{\text{vic}}$ and $\mathbf{w}^3_{\text{vic}}$ become the first ($i'=1$) and second ($i'=2$) vectors, respectively. The directions of all three vectors remain the same. D multiplies one constant scalar s to each $\mathbf{w}^i_{\text{vic}}$ and also does not modify the vector direction. After applying D, all vectors in the left part of Fig. 4 become shorter and the directions do not change.

Difficulty of Changing Direction. We demonstrate that only using per-vector operation without matrix multiplication (as lightweight obfuscation algorithms do), it is difficult to design an algorithm that can change the direction. Our conclusion is that, if the defender can obfuscate the directions, the defender must perform a matrix multiplication to recover Y_{vic}. The only condition that TEE does not need to perform matrix multiplication is that the directions of W_{vic} and W_{obf} are the same.

5 ArrowMatch

We introduce ArrowMatch, a novel direction similarity attack against weight obfuscation algorithms in TSLP. Fig. 5 shows the pipeline of ArrowMatch. ArrowMatch takes M_{obf} and M_{pre} as inputs and outputs the surrogate model M_{sur} which has a similar functionality as M_{vic}. The attack process consists of two steps: distance-based direction recovery (**S1**) and learning-based length adjustment (**S2**).

The goal of **S1** is to use direction similarity to inverse the permutation operation. Based on the attack insight, **S1** iterates all vector pairs between W_{pre} and W_{obf}, computing the direction distance within each pair, and select the pair with the smallest distance. **S1** aims to identify an index mapping $\sigma(\cdot)$, which is the reverse function of $\pi(\cdot)$ and satisfies $\pi(\sigma(i)) = i$. $\sigma(\cdot)$ maps the vector $\mathbf{w}^i_{\text{obf}}$ to the original position $\mathbf{w}^{\sigma(i)}_{\text{vic}}$. We use cosine distance ($\cos(\cdot,\cdot)$) to measure the direction similarity. The attacker iterates each $\mathbf{w}^i_{\text{obf}} \in W_{\text{obf}}$ and selects the j-th vector with the lowest distance for the element vectors:

$$\sigma(i) = \arg\min_j \cos(\mathbf{w}^i_{\text{obf}}, \mathbf{w}^j_{\text{pre}}). \tag{1}$$

For example, in Fig. 5, during the obfuscation phase, $\mathbf{w}^1_{\text{vic}}$ is mapped to the fourth position in W_{obf}, thus $\pi(1) = 4$. The attacker identifies the reverse mapping $\sigma(4) = 1$ and recovers the vector index.

The goal of **S2** is to recover the obfuscation scalar that is multiplied to $\mathbf{w}^i_{\text{vic}}$. We first compare the length between $\mathbf{w}^i_{\text{obf}}$ and $\mathbf{w}^{\sigma(i)}_{\text{pre}}$ to recover an approximation of s_i, denoted as $\hat{s}_i$. Let $l(\cdot)$ be a length function for a vector. The approximation of s_i is computed as $\hat{s}_i = l(\mathbf{w}^{\sigma(i)}_{\text{pre}})/l(\mathbf{w}^i_{\text{obf}})$. We then use $\hat{s}_i$ to initialize the approximate vector $\mathbf{w}^{\sigma(i)}_{\text{init}} = \hat{s}_i \cdot \mathbf{w}^i_{\text{obf}}$ and construct the

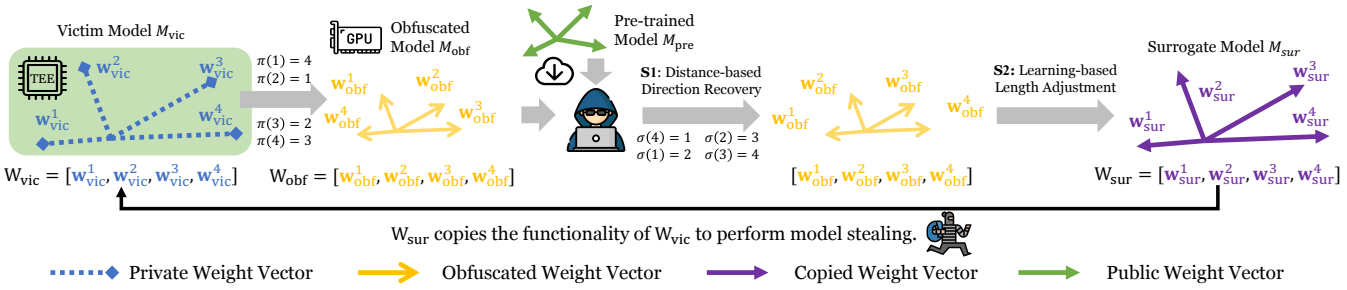

Figure 5: The attack pipeline of ARROWMATCH. The attacker uses M_{vic} and M_{pre} to perform the attack. The attack consists of two steps: distance-based direction recovery (**S1**) and learning-based length adjustment (**S2**).

initialization weight matrix as $W_{\text{init}} = [\mathbf{w}^{\sigma(1)}_{\text{init}}, \mathbf{w}^{\sigma(2)}_{\text{init}}, \ldots, \mathbf{w}^{\sigma(n)}_{\text{init}}]$. We construct W_{init} for each layer to get the initialized model M_{init}. We then train M_{init} with little budget (training data and iterations) to get M_{sur}. As shown in Fig. 5, the attacker can recover the vector length to a similar value as W_{vic} after **S2**.

6 ARROWCLOAK

In this section, we present ARROWCLOAK, a novel lightweight obfuscation algorithm that introduces attacker-invisible obfuscation vectors to safeguard the direction of model vectors. We introduce the insight (Sec.6.1) and the pipeline of ARROWCLOAK (Sec.6.2). Finally, we analyze the security of ARROWCLOAK (Sec.6.3) by reducing the obfuscation algorithm to the LWE problem.

6.1 Defense Design

Fundamental Limitation. The fundamental limitation of existing lightweight obfuscation algorithms is that *they only use per-vector operations and fail to fully exploit the TEE's $O(nl)$-complexity operations to design obfuscation algorithms*. Prior work mainly focuses on per-vector obfuscation (e.g., scaling, shuffling, and addition) because such operations are straightforward to consider low-complexity operations. However, these operations are insufficient to protect the direction of the private vectors in front of ARROWMATCH. According to our analysis in **Obs2**, per-vector operations can not modify the direction of the private vectors. This allows attackers to recover the direction of private vectors from the obfuscated weights, serving as a starting point for efficient MS attacks.

Our Insight: Matrix-Vector Multiplication. To address this limitation, our insight is to *introduce a new low-complexity operation, matrix-vector multiplication, to design the obfuscation algorithm*. Prior work regards matrix-vector multiplication as one type of matrix multiplication and does not consider it executable in TEEs. However, we find that matrix-vector multiplication can be executed in TEEs with $O(nl)$ complexity. For matrix $X \in \mathbb{R}^{l\times n}$ and vector $v \in \mathbb{R}^n$, the vector multiplication operation $X \cdot v$ has a nl complexity and is affordable

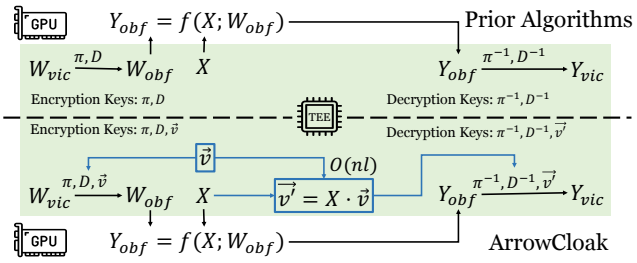

Figure 6: The difference between the existing obfuscation algorithms and ARROWCLOAK. We highlight the difference of ARROWCLOAK in blue squares and lines. ARROWCLOAK uses a new $O(nl)$-complexity operation, matrix-vector multiplication, in TEE to obfuscate the vector direction of W_{vic}.

in TEEs. With this operation, we can introduce a new obfuscation vector $\mathbf{v}$ to protect the direction of the private weight vectors. Figure 6 illustrates the comparison between prior obfuscation algorithms and ARROWCLOAK. The difference of ARROWCLOAK is highlighted in blue squares and lines. As we can see, prior work mainly uses Π and D to obfuscate W_{vic}. In contrast, ARROWCLOAK performs a matrix-vector multiplication ($\mathbf{v}' = X \cdot \mathbf{v}$) to obfuscate W_{vic} and recover Y_{vic} in TEE.

ARROWCLOAK Design We construct $\mathbf{v}$ as the linear combination of the column vectors of W_{vic}, i.e., $\mathbf{v} = \sum k_i \cdot \mathbf{w}^i_{\text{vic}}$. We then obfuscate W_{vic} by scaling and adding $\mathbf{v}$ to each column vector $\mathbf{w}^i_{\text{obf}} = p_i \cdot \mathbf{w}^i_{\text{vic}} + q_i \cdot \mathbf{v}$. Note that k_i, p_i, and q_i are drawn from $\mathbb{Z}_u$, where u is a large prime number. Finally, we shuffle the column vectors as prior work does. The obfuscation process of ARROWCLOAK can be formulated as:

$$W_{\text{obf}} = [p_i \cdot \mathbf{w}_i + q_i \cdot \mathbf{v}] \cdot \Pi = (W_{\text{vic}} \cdot D_1 + \mathbf{v} \cdot \mathbf{1}_n \cdot D_2) \cdot \Pi. \quad (2)$$

where $\mathbf{v} \in \mathbb{R}^{n\times 1}$,$\mathbf{1}_n$ is a vector of ones and $\mathbf{v} \cdot \mathbf{1}_n$ is a matrix with $\mathbf{v}$ as each column. Note that $\mathbf{v} \cdot \mathbf{1}_n$ is a $O(nl)$ complexity operation because it can be implemented by copying $\mathbf{v}$ for n times. D_1 and D_2 are diagonal matrices as follows:

$$D_1 = \text{diag}(p_1, \cdots, p_n), \quad D_2 = \text{diag}(q_1, \cdots, q_n). \quad (3)$$

The offloaded computation can be expressed as:

$$Y_{\text{obf}} = X \cdot W_{\text{obf}} = X \cdot (W_{\text{vic}} \cdot D_1 + \mathbf{v} \cdot \mathbf{1}_n \cdot D_2) \cdot \Pi. \quad (4)$$

Thus we have:

$$Y_{\text{obf}} - X \cdot \mathbf{v} \cdot \mathbf{1}_n \cdot D_2 \cdot \Pi = X \cdot W_{\text{vic}} \cdot D_1 \cdot \Pi. \quad (5)$$

The recovery process in TEE can be formulated as:

$$\begin{aligned} Y_{\text{vic}} &= X \cdot W_{\text{vic}} = X \cdot W_{\text{vic}} \cdot D_1 \cdot \Pi \cdot \Pi^{-1} \cdot D_1^{-1} && (6)\\ &= (Y_{\text{obf}} - X \cdot \mathbf{v} \cdot \mathbf{1}_n \cdot D_2 \cdot \Pi) \cdot \Pi^{-1} \cdot D_1^{-1} && (7)\\ &= Y_{\text{obf}} \cdot \Pi^{-1} \cdot D_1^{-1} - \underbrace{X \cdot \mathbf{v}}_{O(nl)} \cdot \mathbf{1}_n \cdot D_2 \cdot D_1^{-1}. && (8) \end{aligned}$$

Computation Complexity Analysis. The matrix-vector multiplication $X \cdot \mathbf{v}$ has a $O(nl)$ complexity, and multiplying it by $\mathbf{1}_n$ is $O(nl)$ as well. The permutation operation (Π^{-1}) and scaling operations (D_1^{-1} and $D_2 \cdot D_1^{-1}$) are $O(nl)$ complexity. Thus, the recovery process has a $O(nl)$ complexity, which is efficient for TEEs. Note that ArrowCloak relies on three encryption keys to protect W_{vic}: the linear combination coefficients k_i, the scaling coefficients p_i and q_i, and the permutation matrix Π. Similar to prior work [52, 58, 82], these keys are stored in TEEs and can be periodically updated to enhance security.

Protecting `Op_quadra`. ArrowCloak can also protect `Op_quadra` with the matrix-vector multiplication. This way, the adversary can not utilize the internal features in `Op_quadra` to steal the model's privacy. We leave the detail of applying ArrowCloak to `Op_quadra` in Appx. B.

6.2 ArrowCloak Pipeline

ArrowCloak consists of two phases: an offline model obfuscation phase and an online heterogeneous inference phase. The first stage is to obfuscate the model weights and generate W_{obf}, so that W_{vic} can be safely deployed to the untrusted edge devices. The second stage uses W_{obf} to perform inference in the edge devices and recover the true output in the TEE. The inference process is iteratively conducted from the first layer to the last layer.

Offline Model Obfuscation. This phase obfuscates model weights in TEE or the model owner's server. The obfuscation process is conducted as introduced in Sec. 6.1 in a layer-wise manner. The obfuscated weights W_{obf} are generated and stored in the TEE. The obfuscation keys (k_i, p_i, q_i, and Π) are also stored in the TEE. After a fixed time period, the model owner updates the obfuscation keys to enhance security.

Online Heterogeneous Inference. This phase is conducted for each input sample and iteratively computed for each layer. For the non-polynomial layers (`Op_non-poly`; e.g., LayerNorm), the TEE directly computes the results. For the linear layers (`Op_linear`), ArrowCloak offloads the computation to untrusted GPUs with the obfuscated weights W_{obf}. For each `Op_linear`, the offload computation consists of three steps: TEE preparation, GPU computation, and TEE recovery. The first step sends X and W_{obf} to the GPU. The second step computes the obfuscated results Y_{obf} in the GPU. This step is the same as normal GPU computation. The third step follows Eq. 8 to recover the true output Y_{vic} in the TEE. For the integrity check and protection of X and Y_{obf}, we follow prior work to use Freivalds' algorithm [17] and One-Time-Pad [1] as introduced in Sec. 2.3.

6.3 Security Analysis

Learning with Error (LWE). Learning with Errors (LWE) is a computational problem closely related to lattice-based cryptography [48]. Its main claim to fame is that it is known to share the same complexity of worst-case lattice problems, which provide well-learned provable security [49]. The LWE problem involves solving a noisy linear system where random errors are intentionally added to the equations. Due to these carefully chosen errors, it is difficult to recover the original solution (secret) of perturbed equations. The LWE problem has already found many applications in the crypto-world, including homomorphic encryption [7], digital signatures [15], and secure key exchange [5].

LWE Definition (in matrix representation). Let $\mathbb{Z}_u$ be a finite integer field where u is a big prime number. $\mathbf{s} \in \mathbb{Z}_u^n$ is a n-dimension secret vector, and $S = [\mathbf{s}^i]$ is a secret matrix. $A, B \in \mathbb{Z}_u^{m \times n}$ are two public matrices. E is a random error matrix where $e \in E$ is drawn from a gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The standard deviation σ is larger than $2\sqrt{n}$ to prevent attackers from recovering the secret matrix S. The LWE problem can be formulated as:

$$B = A^\top \cdot S + E \mod q,\ A, B \in \mathbb{Z}_u^{m \times n},\ S, E \in \mathbb{Z}_u^{n \times n}. \quad (9)$$

Even if the attacker can learn A and B, it is hard to recover the secret matrix S from the LWE problem.

Reducing to LWE. We present the formulation to reduce the obfuscation algorithm to the LWE problem. According to Eq. 2, $\mathbf{w}_{\text{obf}}^i$ can be represented as:

$$\begin{aligned} \mathbf{w}_{\text{obf}}^i &= p_i \cdot \mathbf{w}_i + q_i \cdot \mathbf{v} = p_i \cdot \mathbf{w}_i + q_i \cdot \sum_{j=0}^{n} (k_j \cdot \mathbf{w}_j) \\ &= (p_i + q_i \cdot k_i) \cdot \mathbf{w}_i + q_i \cdot \sum_{j \neq i} (k_j \cdot \mathbf{w}_j) \quad (10)\\ &= [\mathbf{w}_1, \ldots, \mathbf{w}_n] \cdot [q_1 \cdot k_1, \ldots, p_i + q_i \cdot k_i, \ldots, q_n \cdot k_n]^\top. \end{aligned}$$

W_{obf} can be represented as a matrix multiplication form as:

$$W_{\text{obf}} = W_{\text{vic}} \cdot H \cdot \Pi,\ H = \begin{bmatrix} p_1 + q_1 \cdot k_1 & \cdots & q_n \cdot k_n \\ \vdots & \ddots & \vdots \\ q_1 \cdot k_n & \cdots & p_n + q_n \cdot k_n \end{bmatrix}. \quad (11)$$

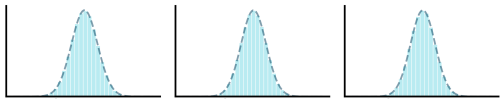

Figure 7: E's distribution for three layers(layer1, layer6 and layer12) of BERT-Base.

Let $\tilde{H} = H \cdot \Pi$, we have $W_{\text{vic}} = W_{\text{obf}} \cdot \tilde{H}^{-1}$. By decomposing W_{pre} into W_{vic} and a residual matrix, we have:

$$\underbrace{W_{\text{pre}}}_{\hat{B}} = W_{\text{vic}} + (W_{\text{pre}} - W_{\text{vic}}) = \underbrace{W_{\text{obf}}}_{\hat{A}^{\top}} \cdot \underbrace{\tilde{H}^{-1}}_{\hat{S}} + \underbrace{(W_{\text{pre}} - W_{\text{vic}})}_{\hat{E}}. \tag{12}$$

Here $\hat{A}$, $\hat{B}$, $\hat{S}$, and $\hat{E}$ are four matrices of real numbers. After quantizing the real numbers to a finite field $\mathbb{Z}_u$ (m is the quantization factor) by

$$A = \lfloor \hat{A} \cdot 2^m \rceil,\ B = \lfloor \hat{B} \cdot 2^m \rceil,\ S = \lfloor \hat{S} \cdot 2^m \rceil,\ E = \lfloor \hat{E} \cdot 2^m \rceil, \tag{13}$$

we can reduce Eq. 11 to the LWE problem Eq. 9.

Error Matrix Distribution. The distribution of error E is crucial to the security of LWE based schema [48]. E should be drawn from a discrete Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, and its standard deviation σ should be larger than $2\sqrt{n}$. We validate these requirements in ArrowCloak through both quantitative and qualitative experiments. We first plot the distribution of $W_{\text{pre}} - W_{\text{vic}}$ in each model. Fig. 7 shows that the distribution of three randomly selected layers of BERT-Based. From these figures, we can qualitatively observe that $W_{\text{pre}} - W_{\text{vic}}$ is close to a normal distribution. We then use the Kolmogorov-Smirnov test to perform hypothesis tests to validate the normality of $W_{\text{pre}} - W_{\text{vic}}$. The results show that the p-values are larger than 0.05 in all methods, which means that there is no significant evidence to reject the hypothesis that $W_{\text{pre}} - W_{\text{vic}}$ is normally distributed. Finally, we compute σ of $W_{\text{pre}} - W_{\text{vic}}$. Note that for LLMs, $n = 768$, and the σ threshold is $2\sqrt{768} \approx 55$. For all models, the deviations are larger than the threshold, which matches the requirement of the LWE problem.

7 Evaluation

In this section, we evaluate the performance of ArrowMatch and ArrowCloak. We first illustrate the evaluation setup and answer the following research questions.

RQ1: How is the performance of ArrowMatch?
RQ2: How is the attack performance in identifying vector matches and lengths?
RQ3: How is ArrowCloak's defense effectiveness against ArrowMatch?
RQ4: How much can ArrowCloak protect the vector direction of model weights?
RQ5: What is the performance of ArrowCloak on real-world TEE devices?
RQ6: Is ArrowMatch scalable in different fine-tune scenarios?

7.1 Evaluation Setup

Models. We evaluate four representative models: BERT-Base [13], GPT2-Base [44], ViT-Base [14], and GPT2-XL [44]. BERT-Base and GPT2-Base are selected from the auto-encoding and auto-regressive architectures, respectively. They are the default models in their respective papers [51]. GPT2-Base has a similar architecture to modern LLMs such as OpenAI's ChatGPT, Meta's LLaMA, and Google's PaLM [47]. ViT-Base is one of the default models in the vision transformer architecture. We select GPT2-XL as a larger model to demonstrate the versatility of ArrowMatch. GPT2-XL has 1.5B parameters, which is similar to modern state-of-the-art edge models such as Phi [2].
Datasets. We select seven representative datasets for each model following prior work. For ViT-Base, we select CIFAR10, CIFAR100 [28], and Food101 [6] following recent TSLP work [31, 82, 83]. For BERT and GPT, we select four datasets from the GLUE benchmark [65]: MNLI [74], QQP [27], SST-2 [54], and QNLI [45]. We select the GLUE benchmark because it is a default benchmark to evaluate the model's performance on text tasks [12, 13, 34]. We follow recent work to select the four datasets [31] because they are larger and more complex than other datasets [65]. For all datasets, we follow the default train/test split. We use the train split to train M_{vic} and use the test split to evaluate.
Defenses. We select five representative lightweight obfuscation algorithms from Tab. 1: SOTER [52], TSQP [57], TransLinkGuard [31], Tempo [76], and ShadowNet [82]. We do not select GroupCover because it has high computation overhead. We do not select KV-Shield [78] and CoreGuard [32] because their obfuscation algorithms are the same as TransLinkGuard. We follow the original paper to implement and set hyper-parameters for each defense. For SOTER, we put 20% of weights in TEE and obfuscate the remaining weights following the original paper. For TSQP, we follow the paper to train the dissimilar model M_{vic} [57]. TSQP has two hyper-parameters for the loss function, α and β, to make M_{vic} dissimilar to M_{pre}. We perform a grid search to find the optimal setting to satisfy the accuracy requirement. For the scaling techniques in SOTER, ShadowNet and Tempo, we follow the settings in prior works [82, 83].

Baselines. We compare the attack performance with three baselines following prior work: No-Shield, Shield-Whole, and M_{obf}-Based [31, 57, 82, 83]. No-Shield means the defender offloads the whole model to the untrusted GPU and does not use TEE. The adversary performs a white-box attack because the entire model is exposed. Shield-Whole means the defender shields the whole model in TEE and does not offload to GPU. The adversary performs a black-box attack because he can not access private weights of M_{vic}. Note that under our threat model (Sec. 3), No-Shield is the security lower bound and Shield-Whole is the security upper bound. M_{obf}-Based means the attacker uses the obfuscated model weights W_{obf} as W_{init} to train M_{sur} and the attacker does not use our attack. This is a common attack used by prior literature to demonstrate the effectiveness of lightweight obfuscation algorithms [31,57,76]. We include this baseline to validate the correctness of our defense implementation and demonstrate the effectiveness of our attack.

Other Settings. We download M_{pre} from public timm [73] and transformers [75] libraries on the Internet. Both libraries are widely used during the development of LLM [10, 42, 61]. We first train M_{vic} based on M_{pre}. The hyper-parameters (the learning rate and the number of epochs) for training follow prior work and public code snippets [81]. For ViT, we set the learning rate to 1e-3, and train for 10 epochs [14]. For BERT-Base and GPT2-Base, we set the learning rate to a range of 1e-5 to 3e-5 and train for 3 epochs [8, 37, 43]. We also follow prior work [29,31,82] to set the dataset size that the adversary can access. We set the size to be less than 1% of the training dataset for each task.

Metric. We follow prior work to report two metrics: the accuracy of M_{sur} [41] and the relative performance compared to the black-box baseline [83]. A higher accuracy represents that M_{sur} steals more functionality of M_{vic} and represents a higher attack performance. We report relative performance to compare across different defense schemes and datasets.

7.2 Attack Performance

We display the attack performance of ArrowMatch in Tab. 3. The last row represents the average performance across all models and datasets. We can observe that ArrowMatch can effectively recover the lightweight obfuscation algorithms and achieve a high attack performance. The attack performance is averagely 1.67× higher than the black-box baseline, while the white-box upper bound of No-Shield is 1.70×. The performance of ArrowMatch is similar to white-box. It means that ArrowMatch can recover the knowledge of the obfuscated weights and improve the attack efficiency.

On the contrary, the M_{obf}-Based baseline has a much lower attack performance (average 0.64× than the black-box baseline), even lower than Shield-Whole (1×). We can also observe that the performance of M_{obf}-Based is nearly the random guess. It is because the obfuscated weights W_{obf} give a very poor initialization for M_{sur}, this initialization is even worse than not using any exposed knowledge. Thus W_{obf} can not be used to train a surrogate model. It means that existing lightweight obfuscation algorithms are effective in front of the naive M_{obf}-Based attack. But ArrowMatch can utilize the direction similarity to break these defenses and achieve a high attack performance.

For the defenses, we can observe that the SOTER and TSQP have better protection than the other three defenses (lower attack performance). It is because SOTER deploys a portion (20%) of linear layers in TEE and reduces the exposure of private weights. However, SOTER also increases the computational overhead of the TEE by a large margin. TSQP modifies the training process and adds a loss term to reduce the similarity between W_{vic} and W_{pre}. But to maintain high accuracy, the weight of this loss term can not be too large, which limits the protection of TSQP. This phenomenon aligns with **Obs1** that W_{vic} can not deviate too much from W_{pre} to maintain high accuracy. For the other three defenses, they do not modify the training process and shield more layers, thus they are more vulnerable to our attack.

Answer to RQ1: ArrowMatch can effectively recover the lightweight obfuscation algorithms and achieve a high attack performance. The attack performance is averagely 1.67× higher than the black-box baseline (white-box upper bound is 1.70×).

7.3 Attack Analysis

In this section, we analyze the performance of the two steps in our attack pipeline (**S1** and **S2**).

Recovered Permutation. We first evaluate the performance of **S1**. We use the recovered accuracy of the permutation index to evaluate the performance. The accuracy is computed as the percentage of the correct recovered permutation index: $Acc_{\sigma} = \frac{1}{N}\sum_{i=1}^{N}[\sigma(\pi(i)) = i]$, where N is the total number of vectors in the model. A higher accuracy means a better recovery performance. We display the accuracy for all models and datasets in Tab. 4. We can observe that the accuracy is very high (over 99%). It means that the attacker can effectively recover the permutation index and identify the mapping between the obfuscated vectors and the original vectors. This high accuracy is reasonable given the validation of **Obs1** (Tab. 2) that the distance of `Original` is averagely 78× smaller than the distance of `Second-Closest`.

Recovered Vector Length. We then evaluate the performance of **S2**. We use the similarity between the vector lengths of M_{sur}'s linear layers and the corresponding vector lengths of M_{vic} to evaluate the performance. The similarity is computed as the average percentage of length difference over the length of M_{vic}: $Sim = \frac{1}{N}\sum_{i=1}^{N} |\|\mathbf{w}^i_{sur}\| - \|\mathbf{w}^i_{vic}\||/\|\mathbf{w}^i_{vic}\|$. Tab. 4 shows the results. We can observe that the similarity is very high (over 98%), which means the attacker can ef-

Table 3: The attack performance against lightweight obfuscation algorithms. We report the accuracy of M_{vic} to measure the MS attack. "C10", "C100", "F101" represent CIFAR10, CIFAR100, and Food101, respectively. The last row reports the average accuracy toward each defense relative to the baseline black-box Shield-Whole solution.

		SOTER		TSQP		TransLinkGuard		Tempo		ShadowNet		White-box	Black-box
		M_{obf}-Based	Our Attack	M_{obf}-Based	Our Attack	M_{obf}-Based	Our Attack	M_{obf}-Based	Our Attack	M_{obf}-Based	Our Attack		
ViT-Base	C10	9.58%	98.35%	11.84%	98.01%	16.87%	98.56%	12.95%	98.53%	12.92%	98.54%	98.84%	51.92%
	C100	2.35%	87.77%	2.08%	86.77%	5.37%	89.89%	2.77%	89.86%	1.22%	89.50%	92.76%	44.08%
	F101	1.47%	83.38%	1.41%	81.83%	2.38%	85.56%	1.39%	84.78%	1.16%	84.61%	91.49%	36.25%
BERT-Base	MNLI	32.90%	83.16%	33.51%	82.73%	32.74%	83.58%	33.26%	83.48%	31.95%	83.43%	83.91%	56.48%
	QQP	46.12%	89.92%	37.67%	87.01%	63.81%	90.67%	36.81%	90.53%	63.18%	90.52%	90.80%	75.36%
	SST-2	51.61%	90.37%	51.15%	91.06%	50.92%	91.74%	49.20%	91.28%	50.92%	91.40%	91.86%	55.28%
	QNLI	49.31%	91.03%	49.59%	90.90%	49.44%	90.87%	49.46%	91.01%	50.54%	90.92%	90.61%	70.60%
GPT2-Base	MNLI	32.79%	79.93%	32.74%	73.29%	33.96%	81.15%	32.71%	81.07%	33.39%	81.08%	81.15%	42.64%
	QQP	55.06%	86.02%	62.67%	86.33%	64.47%	86.86%	59.85%	86.85%	61.95%	86.86%	87.27%	70.89%
	SST-2	48.74%	89.45%	48.39%	90.02%	50.92%	90.14%	50.00%	89.91%	49.43%	89.91%	91.28%	48.85%
	QNLI	49.22%	85.08%	49.44%	85.76%	52.65%	85.36%	48.65%	85.70%	51.16%	85.67%	86.69%	46.66%
GPT2-XL	SST-2	49.31%	94.72%	49.43%	95.18%	50.92%	94.95%	48.51%	94.95%	49.08%	94.95%	94.95%	66.97%
Average		0.62×	1.68×	0.62×	1.64×	0.68×	1.68×	0.62×	1.68×	0.65×	1.67×	1.70×	1.00 ×

fectively recover the vector length using a small amount of training data. The vector length of M_{sur} is very close to the vector length of M_{vic}. The high similarity is due to the good initialization of vector position in **S1**.

Table 4: The recovery accuracy on permutation (**S1**) and vector length (**S2**).

	Permutation (**S1**)				Length (**S2**)			
	C10	C100		F101	C10	C100		F101
ViT-Base	99.99%	99.99%		99.98%	98.80%	98.81%		98.19%
	MNLI	QQP	SST2	QNLI	MNLI	QQP	SST2	QNLI
BERT-Base	100%	100%	100%	100%	99.90%	99.79%	99.98%	99.94%
GPT2-Base	100%	100%	100%	100%	99.99%	99.99%	99.99%	99.99%

Answer to RQ2: Both steps are effective in our attack and can achieve high accuracies of over 98%

7.4 Defense Effectiveness

In this RQ, we compare the defense effectiveness with prior obfuscation methods. We use the same attack and hyper-parameter setup as in Sec. 7.1. Fig. 5 shows the attack performance against ArrowCloak and the best performance of the existing lightweight obfuscation algorithms (Denoted as "Prior Best"). We report M_{sur}'s accuracy and the relative accuracy compared to the performance of Shield-Whole baseline (Denoted as "Rel.Black"). Note that No-Shield puts all the models in TEE and is the security upper bound. Tab. 5 shows the defense effectiveness. We can observe that the attack against ArrowCloak is much lower than the best prior defense. Averagely, the "Rel.Black" of ArrowCloak is 1.10×, but "Prior Best" is 1.65×. ArrowCloak reduces privacy leakage by 6.5× over the best prior obfuscation algorithms. Also, ArrowCloak only increases the attack performance by 10% compared to Shield-Whole, which is very close to the ideal protection. Notably, ArrowCloak performs better for the larger model GPT2-XL, which achieves a "Rel.ArrowCloak" of 0.90×. This result demonstrates that ArrowCloak effectively protects the model privacy against the model stealing attack.

Table 5: The defense performance of ArrowCloak. We compare ArrowCloak with the best performance of prior obfuscation algorithms in Tab. 3 ("Prior Best").

Model	Dataset	ArrowCloak		Prior Best	
		M_{sur}	Rel.Black	M_{sur}	Rel.Black
ViT-Base	C10	68.53%	1.32×	98.01%	1.89×
	C100	55.23%	1.25×	86.77%	2.10×
	F101	39.60%	1.09×	81.83%	2.26×
BERT-Base	MNLI	57.04%	1.01×	82.73%	1.46×
	QQP	74.01%	0.98×	87.01%	1.15×
	SST2	73.51%	1.33×	90.37%	1.63×
	QNLI	63.44%	0.90×	90.90%	1.29×
GPT2-Base	MNLI	48.85%	1.15×	73.29%	1.72×
	QQP	74.46%	1.05×	86.02%	1.21×
	SST2	52.87%	1.08×	89.45%	1.83×
	QNLI	52.30%	1.12×	85.08%	1.82×
GPT2-XL	SST2	60.21%	0.90×	94.72%	1.41×
Average		60.00%	1.10×	87.18%	1.65×

Different Sizes of Adversary's Training Datasets. We also evaluate the defense effectiveness under different sizes of adversary's training datasets. We iterate M_{sur}'s size from 1% to 50% of M_{vic}'s size. Fig. 8 shows the comparison between ArrowCloak (left) and the Shield-Whole baseline (right).

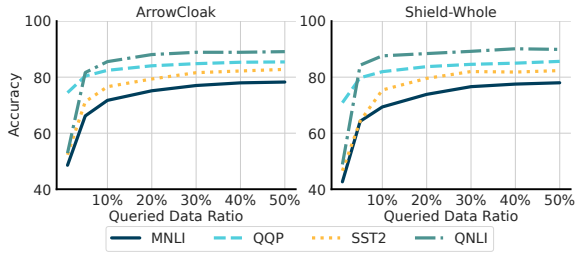

Figure 8: Comparison of ARROWCLOAK and the black-box protection under M_{sur}'s training dataset size.

We can observe that ARROWCLOAK is comparable to Shield-Whole over different dataset sizes. Averagely, the attack accuracy against ARROWCLOAK is 1.10× to Shield-Whole.

Answer to RQ3: ARROWCLOAK can effectively protect the model against ARROWMATCH. The attack performance is only 1.10× higher than Shield-Whole and is 6.5× better than the best prior obfuscation algorithms.

7.5 Vector Direction Protection

The main insight of ARROWMATCH is that prior lightweight obfuscation algorithms do not protect vector directions of model weights. In this RQ, we study ARROWCLOAK's effectiveness in protecting directions. We measure the direction similarity between three types of vector matches: `Prior Obf`, `Our Obf`, and `Random`. `Prior Obf` represents correct vector match between other obfuscation's M_{obf} and M_{pre}, `Our Obf` represents correct vector match between ARROWCLOAK's M_{obf} and M_{pre}, and `Random` represents a randomly selected vector match beween M_{vic} and M_{pre}. Remember that in Sec. 4.1, due to **Obs1** (low direction discrepancy) and **Obs2** (direction invariance), the distance of correct matches of other obfuscation algorithms (`Prior Obf`) is low. We regard `Random` is a distance upper bound because the best direction obfuscation algorithm is to make the obfuscated vector randomly distributed. Fig. 9 shows the comparison over three distance functions (cos, L_2, and L_∞). The blue, green, and yellow bars represent `Prior Obf`, `Our Obf`, and `Random`, respectively. We can observe that the distance of `Our Obf` is close to `Random` (averagely 0.91×). On the contrary, the distance of `Prior Obf` is much lower than `Random` (averagely 0.01×). `Our Obf` has an averagely 961.94× larger distance than `Prior Obf`. These results demonstrate that ARROWCLOAK effectively protects the vector direction of the model weights, which is a significant improvement over prior obfuscation algorithms.

Answer to RQ4: ARROWCLOAK can effectively protect the vector direction in W_{vic}. The distance after obfuscation is similar to a random vector and is 961.94× larger than prior obfuscation algorithms.

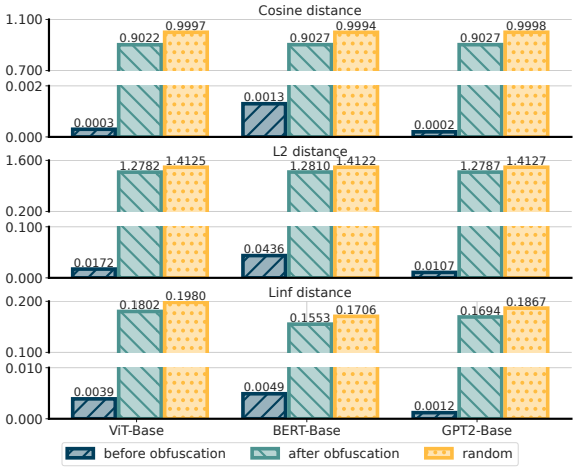

Figure 9: The comparison on the vector direction protection. We report the performance of prior obfuscation algorithms (`Prior Obf`; blue bar), ARROWCLOAK (`Our Obf`; green bar), and a random vector (`Random`; yellow bar). We report three distance functions: cos, L_2, and L_∞.

7.6 Real-Device Performance

Setup. To evaluate the performance of ARROWCLOAK on real-world TEE devices, we implement ARROWCLOAK on the latest public framework, TAOISM [83]. We run the experiments on a testbed with an Intel Core i7-8700 3.20GHz CPU and NVIDIA GeForce GTX 1080 GPU. The framework consists of two components: a shielded part in SGX and an offloaded part in GPU. We implement the ARROWCLOAK algorithm and communication in the shielded part, and reuse the GPU offloaded part. We switch the SGX configuration to the hardware mode to emulate the real-world environment. We run the experiments five times and report the average inference time. We report the average per-sample inference time (ms). All the results are averaged over ten runs, and the deviation is less than 5%.

Baselines. We compare ARROWCLOAK with four baselines: Shield-Whole, ShadowNet, TSQP, and Slalom [62]. Shield-Whole means to shield the whole model in SGX, and we use Shield-Whole to report how much overhead ARROWCLOAK saves. We use ShadowNet and TSQP are two state-of-the-art obfuscation algorithms. Slalom is the first TSLP work that offloads $\texttt{Op}_{\texttt{linear}}$ to GPU and puts other parts in SGX. Although Slalom does not protect model weights, we use Slalom to show how much overhead the ARROWCLOAK's obfuscation algorithm introduces. Because the only difference between Slalom and ARROWCLOAK (as well as ShadowNet and TSQP) is the obfuscation algorithm.

Results. Tab. 6 shows the real-device performance. Note that Bert-Base and GPT2-Base have the same architecture for the transformer block. Thus we merge the results into one column. We take the Slalom as the threshold and report the relative performance compared to the Slalom. Compared to Slalom, ARROWCLOAK has an average of 0.46× additional

Table 6: The performance of ARROWCLOAK on a real SGX device compared with five baselines: Shield-Whole, No-Shield, ShadowNet, TSQP, and Slalom. We report the inference time (ms) and the relative performance over Slalom.

	ViT-Base	GPT2-Base	GPT2-XL
Shield-Whole	778.02(3.42×)	1778.55(3.67×)	23290.46(5.36×)
ARROWCLOAK	323.83(1.42×)	725.29(1.50×)	6394.42(1.47×)
ShadowNet	305.25(1.34×)	669.77(1.38×)	6041.50(1.39×)
TSQP	229.70(1.01×)	493.49(1.02×)	4381.66(1.01×)
Slalom	227.44(1.00×)	485.07(1.00×)	4341.20(1.00×)

Table 7: ARROWCLOAK inference time breakdown.

Data Transfer	GPU Computation	$\texttt{Op}_{\texttt{non-poly}}$ in TEE	Obfuscation Recovery
19.83%	16.31%	23.16%	40.70%

overhead. This overhead is introduced by the obfuscation computation in SGX. Note that ARROWCLOAK's overhead is similar to ShadowNet, which uses Π and D to obfuscate. As displayed in Sec. 7.2 and Sec. 7.4, ARROWCLOAK has a much better defense effectiveness than ShadowNet. Compared to Shield-Whole, ARROWCLOAK can save 2.83× overhead, which demonstrates the effectiveness of ARROWCLOAK's lightweight obfuscation algorithm. TSQP has a small overhead because it only multiplies a scalar at runtime, but TSQP has a much larger training overhead and is vulnerable to vector direction attack (Sec. 7.2). We can also observe that, as the model size increases, the overhead benefits of ARROWCLOAK increase as well. For ViT-Base, ARROWCLOAK outperforms Shield-Whole by 2.40×. For the large model GPT2-XL, ARROWCLOAK outperforms by 3.64×. This is because a larger model uses a larger weight matrix, which can be effectively accelerated in GPUs.

Tab. 7 displays the inference time break done for ARROWCLOAK. The computation in GPU only takes 16.31% of the total time, which means that ARROWCLOAK effectively utilizes GPU's computation capability to accelerate the inference process. Y_{vic} recovery in TEE takes most of the time (40.70%). This is because we need to perform matrix-vector multiplication in TEE to protect the outsourced computation. This multiplication can provide strong security protection. Although ARROWCLOAK (and other TSLP) introduces additional data transfer overhead (19.83%), ARROWCLOAK can still effectively improve the overall performance because of $\texttt{Op}_{\texttt{linear}}$ and $\texttt{Op}_{\texttt{non-poly}}$ offload, which validates the insight of TSLP solutions.

Answer to RQ5: ARROWCLOAK can reduce the overall latency by 2.83× compared to Shield-Whole. Compared to a non-obfuscated TSLP solution (Slalom), ARROWCLOAK only introduces 0.46× overhead.

Table 8: The attack performance of ARROWCLOAK in LoRA scenarios.

	ViT-Base	BERT-Base	GPT2-Base	Average
SOTER	81.95%	89.56%	87.16%	1.66×
TSQP	85.33%	90.36%	87.27%	1.69×
TransLinkGuard	85.26%	90.37%	87.39%	1.70×
Tempo	85.01%	90.36%	87.39%	1.69×
ShadowNet	85.06%	90.36%	87.39%	1.69×
ARROWCLOAK	38.32%	71.67%	51.26%	1.01×
White-box	92.65%	90.37%	88.30%	1.75×
Black-box	47.16%	60.32%	49.20%	1.00×

7.7 Scalability to LoRA Models

Settings. To validate the generalizability of our methods, we conducted experiments using LoRA [24], one of the most popular parameter-efficient fine-tuning methods. We evaluated our approach on three base models: ViT-Base [14], BERT-Base [13], and GPT2-Base [44], with LoRA rank set to 16 while keeping the learning rate and number of epochs consistent with full fine-tuning configurations. We selected CIFAR100 [28] and SST-2 [54] as the experimental datasets to evaluate performance on image and text tasks.

Baselines. We select the same lightweight obfuscation baselines [31, 52, 57, 58, 76] in Tab. 3. The evaluation pipeline consists of four steps. First, the defender initializes the victim model's parameters that need to be fine-tuned. Then, the defender uses LoRA to fine-tune the model on the downstream task. Third, the defender applies the obfuscation methods to protect the model weights. Last, the adversary utilizes ARROWMATCH to attack the obfuscated model.

Results. Tab. 8 shows the attack performance in LoRA scenarios. ARROWMATCH maintains robust effectiveness against all existing lightweight obfuscation techniques. Across all lightweight obfuscation methods, ARROWMATCH provides stable performance (1.65× to 1.70× compared to the black-box baseline), matching the results of full fine-tuning. On the contrary, ARROWCLOAK reduces the attack performance to a similar level of black-box (1.01×). The results demonstrate the scalability of ARROWMATCH and ARROWCLOAK to different fine-tuning scenarios.

Answer to RQ6: ARROWMATCH and ARROWCLOAK are effective in LoRA-based fine-tuning scenario. ARROWMATCH achieves 1.65× to 1.70× performance gain compared with black-box baseline, and ARROWCLOAK can reduce the attack performance to a similar level of black-box baseline.

8 Discussion

Other TSLP Work. Except for obfuscation-based solutions, there are other TSLP solutions. Some early work presumes that offloading the plaintext of unimportant weights to the untrusted GPU is secure [23, 36, 84]. However, a recent study, TEESlice [83], shows that this approach is vulnerable. TEESlice suggests that privacy-irrelevant weights should be offloaded to GPU while privacy-related parts should be shielded [83]. However, TEESlice requires additional training phases. TEESlice is an orthogonal solution to ARROWCLOAK because ARROWCLOAK aims to protect the privacy-related weights on GPU. ARROWCLOAK can be combined with TEESlice to provide more comprehensive protection for LLMs.

GPU TEE. Enabling GPU TEE support is a hot topic in both academia [25, 56, 64, 67] and industry [40]. Industrial level GPU TEE (*e.g.* Nvidia H100 [40]) can be integrated into VM TEEs, such as Intel TDX and AMD SEV, to provide confidential computation. However, most solutions require heavy modification concerning GPU hardware and firmware, which is not applicable to existing commercial GPUs. The GPU TEE is not widely available on commercial products [71]. In contrast, ARROWCLOAK is an algorithm-based solution that can be applied to existing GPUs without changing the hardware. ARROWCLOAK can be seen as a complementary solution to GPU TEEs for production scenarios.

Side-Channel Attacks against TEE. One important threat to TEE is side-channel attacks. Recently, researchers have proposed various side-channel attacks against TEEs to attack TEE-shielded DNN models [79, 80]. Researchers also proposed several mitigation techniques against these side-channel attacks. For example, memory randomization schemes [30, 72] and profiling-based obfuscation schemes [21] can minimize side-channel leakage. ARROWCLOAK can be combined with these side-channel mitigation techniques to provide more comprehensive protection for DNN models in TEEs.

Scalability to Other Models. This paper focuses on the security of LLM on users' devices. In Sec. 7, we demonstrate the effectiveness of ARROWMATCH and ARROWCLOAK on different LLMs (up to 1.5B) and datasets. We believe that ARROWMATCH and ARROWCLOAK are scalable and can be generalized to protect other LLMs. The design of ARROWMATCH and ARROWCLOAK does not rely on specific TEE technique or hardware, making them applicable among different TEE platforms, such as ARM TrustZone [4, 9].

Cloud Data Protection. ARROWCLOAK can also be applied to cloud-based LLMs to protect uploaded user data. The threat model for cloud-based LLMs is described in [62]. In this scenario, the cloud server is semi-honest and can access the model weights. The adversary's goal is to recover the user data from the cloud. Cloud service providers (on behalf of end-users) can apply ARROWCLOAK to the input data of all $\texttt{Op}_{\texttt{linear}}$ layers. We leave the application of ARROWCLOAK to cloud-based LLMs as future work.

Real-Device Performance. Comparing to Shield-Whole, we have shown that ARROWCLOAK already enjoys a 2.83× acceleration, but we believe that the performance of ARROWCLOAK can be further improved. For example, we can use a pipeline-based design [35] to parallelize the communication (of a prior sample) and TEE computation (of a later sample), thus improving overall throughput. We can also optimize the network architecture [50] to reduce the number of shielded layers. We leave these optimizations as future work.

High-Overhead Method in LoRA Scenario. Unlike lightweight obfuscation methods [31,32,52,57,58,76,78], the high-overhead obfuscation method (GroupCover [82]) employs linear combinations of privacy vectors via matrix multiplication. This incurs substantial computational costs in TEE. This issue persists in LoRA scenarios. Specifically, GroupCover can be formulated as $Y' = X \cdot (W + B \cdot A \cdot P)$ in LoRA settings, where $A \in R^{k \times n}$ and $B \in R^{n \times k}$ are LoRA modules, $P \in R^{n \times n}$ is a grouped obfuscation matrix. In the recovery process, TEE has to calculate $X \cdot B \cdot A = (Y' - X \cdot W) \cdot P^{-1}$, resulting in $O(\frac{n^2 \cdot l}{k})$ computational complexity.

Application Scope. Consistent with prior work [29,57,82,83], this paper posits that adversaries can exploit public information from pre-trained models corresponding to victim models to launch attacks. Building upon this, we propose (1) an effective attack method that leverages vector-directional similarity, and (2) an enhanced obfuscation method with stronger security guarantees for model privacy protection. In scenarios where adversary capabilities are constrained (no available pre-trained model), ARROWCLOAK is still effective, and other lightweight obfuscation approaches may also be viable. One example of such a scenario is that the private model is trained from scratch.

Related Work of Direction Similarity. While prior works [20, 39] have explored directional similarity, ARROWCLOAK differs from other methods that use direction similarity because we compute similarity in the model weight space. Attacks like watermarks [39] and adversarial examples [20] focus on the feature space or the input space.

9 Conclusion

This paper studies the security of TSLP's obfuscation algorithms and identifies a novel vulnerability: the direction similarity. We also propose a corresponding novel attack, ARROWMATCH, to exploit this vulnerability. Furthermore, a novel obfuscation scheme, ARROWCLOAK, is introduced to protect the direction information and defend against ARROWMATCH. We extensively evaluate ARROWMATCH and ARROWCLOAK and demonstrate their effectiveness.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. This work was partly supported by the National Science and Technology Major Project of China (2022ZD0119103) and ByteDance.

Ethics Considerations

We have carefully read the ethics considerations discussions in the conference call for papers, the detailed submission instructions, and the guidelines for ethics documents. This research focuses on the study of deep learning models and Trusted Execution Environments (TEEs) for enhancing secure and efficient computing. The datasets employed in this study were curated from publicly available resources that explicitly permit research use. Similarly, the deep learning models explored in this research are widely used in the academic community, ensuring transparency and reproducibility and further mitigating potential ethical risks. Moreover, this study does not involve experiments on live systems, human participants, or data that could be linked to personally identifiable information (PII). Furthermore, no terms of service were violated, and the research adheres strictly to both the "Respect for Persons" and "Respect for Law and Public Interest" principles outlined in The Menlo Report. In summary, this research poses no significant ethical risks.

Open Science

Aligning with the open science policy, we release our code on [69, 70]. The code repository contains implementations of ARROWMATCH and ARROWCLOAK, along with corresponding scripts, across several model architectures(ViT, BERT, and GPT). By following the instructions in the code repository's README file, people can reproduce the attack and defense performance in our paper.

References

[1] One-time pad. https://en.wikipedia.org/wiki/One-time_pad, 2022.

[2] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[3] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7319–7328.

[4] Tiago Alves. Trustzone: Integrated hardware and software security. *White paper*, 2004.

[5] Joppe Bos, Craig Costello, Leo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1006–1018, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, pages 446–461. Springer, 2014.

[7] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, January 2014. Publisher: Society for Industrial and Applied Mathematics.

[8] Paweł Budzianowski and Ivan Vulić. Hello, it's GPT-2 - how can I help you? towards the use of pretrained language models for task-oriented dialogue systems. In Alexandra Birch, Andrew Finch, Hiroaki Hayashi, Ioannis Konstas, Thang Luong, Graham Neubig, Yusuke Oda, and Katsuhito Sudoh, editors, *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 15–22, Hong Kong, November 2019. Association for Computational Linguistics.

[9] Yifeng Cai, Ziqi Zhang, Jiaping Gui, Bingyan Liu, Xiaoke Zhao, Ruoyu Li, Zhe Li, and Ding Li. Famos: robust privacy-preserving authentication on payment apps via federated multi-modal contrastive learning. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, pages 289–306, 2024.

[10] Joel Castaño, Silverio Martínez-Fernández, Xavier Franch, and Justus Bogner. Analyzing the evolution and maintenance of ml models on hugging face. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pages 607–618, 2024.

[11] Yufei Chen, Chao Shen, Cong Wang, and Yang Zhang. Teacher model fingerprinting attacks against transfer learning. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*.

[12] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[15] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, February 2018.

[16] Muhammad Fawi. Curlora: Stable llm continual fine-tuning and catastrophic forgetting mitigation. *arXiv preprint arXiv:2408.14572*, 2024.

[17] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, 1977.

[18] Balaji Ganesan. Protecting Intellectual Property in the Age of LLM Generative AI Tools. https://www.dataversity.net/protecting-intellectual-property-in-the-age-of-llm-generative-ai-tools/, 2023.

[19] Sorab Ghaswalla. Navigating Uncharted Legal Waters: Copyright Concerns Surrounding Large Language Models. https://sorabg.medium.com/navigating-uncharted-legal-waters-copyright-concerns-surrounding-large-language-models-87d18ddbcbcd, 2023.

[20] Dejian Guan, Dan Liu, and Wentao Zhao. Adversarial detection based on local cosine similarity. In *2022 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 521–525, 2022.

[21] Andrei Homescu, Steven Neisius, Per Larsen, Stefan Brunthaler, and Michael Franz. Profile-guided automated software diversity. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2013, Shenzhen, China, February 23-27, 2013*, pages 23:1–23:11. IEEE Computer Society, 2013.

[22] Eliahu Horwitz, Jonathan Kahana, and Yedid Hoshen. Recovering the pre-fine-tuning weights of generative models. In *International Conference on Machine Learning*, pages 18882–18904. PMLR, 2024.

[23] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Trans. Dependable Secur. Comput.*, 19(6):4270–4284.

[24] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*.

[25] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G. Edward Suh. Guardnn: Secure DNN accelerator for privacy-preserving deep learning. *CoRR*, abs/2008.11632, 2020.

[26] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 4:1–4:6.

[27] Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. First quora dataset release: Question pairs, 2017.

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] Ding Li, Ziqi Zhang, Mengyu Yao, Yifeng Cai, Yao Guo, and Xiangqun Chen. Teeslice: Protecting sensitive neural network models in trusted execution environments when attackers have pre-trained models. *arXiv preprint arXiv:2411.09945*, 2024.

[30] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A systematic look at ciphertext side channels on AMD SEV-SNP. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 337–351. IEEE, 2022.

[31] Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. Translinkguard: Safeguarding transformer models against model stealing in edge deployment. In *Proceedings of the 32nd ACM International Conference on Multimedia, MM 2024, Melbourne, VIC, Australia, 28 October 2024 - 1 November 2024*, pages 3479–3488.

[32] Qinfeng Li, Yangfan Xie, Tianyu Du, Zhiqiang Shen, Zhenghan Qin, Hao Peng, Xinkui Zhao, Xianwei Zhu, Jianwei Yin, and Xuhong Zhang. Coreguard: Safeguarding foundational capabilities of llms against model stealing in edge deployment. *arXiv preprint arXiv:2410.13903*, 2024.

[33] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[35] Ruilong Ma, Xiang Yang, Jingyu Wang, Qi Qi, Haifeng Sun, Jing Wang, Zirui Zhuang, and Jianxin Liao. Hpipe: Large language model pipeline parallelism for long context on heterogeneous cost-effective devices. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 1–9, 2024.

[36] 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 Eyal de Lara, Iqbal Mohomed, Jason Nieh, and Elizabeth M. Belding, editors, *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 161–174.

[37] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning {bert}: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations*, 2021.

[38] Tushar Nayan, Qiming Guo, Mohammed Alduniawi, Marcus Botacin, A. Selcuk Uluagac, and Ruimin Sun. Sok: All you need to know about on-device ML model extraction - the gap between research and practice. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[39] Ehsan Nezhadarya, Z. Jane Wang, and Rabab Kreidieh Ward. Robust image watermarking based on multiscale gradient direction quantization. *Trans. Info. For. Sec.*, 6(4):1200–1213, December 2011.

[40] NVIDIA. NVIDIA H100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/h100/, 2023.

[41] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4954–4963.

[42] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A framework for adapting transformers. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online, October 2020. Association for Computational Linguistics.

[43] Raul Puri, Ryan Spring, Mohammad Shoeybi, Mostofa Patwary, and Bryan Catanzaro. Training question answering models from synthetic data. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5811–5826, November 2020.

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

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

[46] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1157–1174.

[47] Rapid Innovation. Advancements in chatbot interactions with transformer models, 2024.

[48] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*.

[49] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 7(30):11, 2010.

[50] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search:

Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

[51] Shuai Ren, Neeraj Dey, and Fuqian Shi. A comprehensive survey on applications of transformers for deep learning tasks. *arXiv preprint arXiv:2306.07303*, 2023.

[52] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, Fengwei Zhang, and Heming Cui. SOTER: guarding black-box inference for general neural networks at the edge. In Jiri Schindler and Noa Zilberman, editors, *2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 723–738.

[53] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model stealing attacks against inductive graph neural networks. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1175–1192.

[54] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[55] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. ACAI: protecting accelerator execution with arm confidential computing architecture. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*.

[56] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. ACAI: protecting accelerator execution with arm confidential computing architecture. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[57] Yu Sun, Gaojian Xiong, Jianhua Liu, Zheng Liu, and Jian Cui. Tsqp: Safeguarding real-time inference for quantization neural networks on edge devices. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1–1. IEEE Computer Society, 2024.

[58] 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 *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1596–1612.

[59] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1955–1972.

[60] Finbarr Timbers. Where do LLMs spend their FLOPS? https://www.artfintel.com/p/where-do-llms-spend-their-flops?t=, 2024.

[61] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):5314–5321, 2023.

[62] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[64] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on gpus. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 681–696. USENIX Association, 2018.

[65] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, 2018.

[66] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. With great training comes great vulnerability: Practical attacks against transfer learning. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1281–1297.

[67] Chenxu Wang, Yunjie Deng, Zhenyu Ning, Kevin Leach, Jin Li, Shoumeng Yan, Zhengyu He, Jiannong Cao, and Fengwei Zhang. Building a lightweight trusted execution environment for arm gpus. *IEEE Transactions on Dependable and Secure Computing*, 2023.

[68] Chenxu Wang, Fengwei Zhang, Yunjie Deng, Kevin Leach, Jiannong Cao, Zhenyu Ning, Shoumeng Yan, and Zhengyu He. CAGE: complementing arm CCA with GPU extensions. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*.

[69] Pengli Wang. Github link. https://github.com/qsxltss/Game-of-Arrows, 2025.

[70] Pengli Wang. Zenodo link. https://zenodo.org/records/15608291, 2025.

[71] Qifan Wang and David Oswald. Confidential computing on heterogeneous cpu-gpu systems: Survey and future directions. *arXiv preprint arXiv:2408.11601*, 2024.

[72] Jan Wichelmann, Anja Rabich, Anna Pätschke, and Thomas Eisenbarth. Obelix: Mitigating side-channels through dynamic obfuscation. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 4182–4199. IEEE, 2024.

[73] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[74] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, 2018.

[75] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[76] Rongwu Xu and Zhixuan Fang. Tempo: Confidentiality preservation in cloud-based neural network training. In *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*, pages 1–10.

[77] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2003–2020.

[78] Huan Yang, Deyu Zhang, Yudong Zhao, Yuanchun Li, and Yunxin Liu. A first look at efficient and secure on-device LLM inference against KV leakage. In *Proceedings of the 19th Workshop on Mobility in the Evolving Internet Architecture, MobiArch 2024, Washington, DC, USA, November 18-22, 2024*, pages 13–18.

[79] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. Ciphersteal: Stealing input data from tee-shielded neural networks with ciphertext side channels. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 79–79. IEEE Computer Society, 2024.

[80] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. Hypertheft: Thieving model weights from tee-shielded neural networks via ciphertext side channels. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 4346–4360. ACM, 2024.

[81] Zheng Zhang. Groupcover. https://github.com/ZzzzMe/GroupCover, 2024.

[82] Zheng Zhang, Na Wang, Ziqi Zhang, Yao Zhang, Tianyi Zhang, Jianwei Liu, and Ye Wu. Groupcover: A secure, efficient and scalable inference framework for on-device model protection based on tees. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[83] 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 *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 3327–3345.

[84] Tong Zhou, Yukui Luo, Shaolei Ren, and Xiaolin Xu. Nnsplitter: An active defense solution for DNN model via automated weight obfuscation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*.

[85] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes attack: Steal DNN models with lossless inference accuracy. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1973–1988.

A Integrity and Communication Security for TSLP

A.1 Integrity Check.

TSLP uses Freivalds' algorithm [17] to verify the correctness of GPU results. This verification process is efficient because it only requires $O(nl)$ operations, where l is the sequence length of the input. Besides, this verification can be randomly performed on a subset of the output vectors to reduce the computational cost. The attacker cannot predict which output vectors will be verified, so the verification process is secure.

A.2 OTP-based Communication Protection.

To ensure the privacy of the input X to outsourced layers, the Slalom employs a One-Time Pad (OTP) encryption scheme. Let $X \in \mathbb{R}^{l\times n}$ represent the input matrix, and let $\mathbf{R} \in \mathbb{F}^{l\times n}$ be a pseudorandom matrix generated within TEE or at model owner's server. The encrypted input $\tilde{X}$ is computed as $\tilde{X} = X + R$, where the addition is performed element-wise over the finite field $\mathbb{R}$. The matrix R, acting as the OTP, is used only once and is securely generated using a cryptographically secure pseudorandom number generator (PRNG).

The encrypted input $\tilde{X}$ is sent to the untrusted GPU along with the obfuscated weight W_{obf}. GPU performs the linear operation to compute Y_{obf}, send the result back to the TEE. The TEE then deobfuscate the result and get $\tilde{Y} = \tilde{X}W$. Then TEE can compute the true output as $Y = \tilde{Y} - R \cdot W$. and $R \cdot W$ is computed within the TEE or at the model owner's server at the offline stage. This scheme ensures that the untrusted GPU gains no information about X, as R is known only to the TEE.

Computation Analysis. In an OPT-based scheme, TEE only needs to perform element-wise addition and subtraction, which is efficient and incurs negligible overhead. The total computation complexity in TEE is $O(nl)$. It is because the matrix multiplication $R \cdot W$ is precomputed offline and does not require any computation in the online stage.

B Protecting Attention Multiplication

In this section, we provide a detailed explanation of the protection of the attention module in TSLP. We first demonstrate that OTP encryption is unsuitable for protecting the attention module. Then we introduce how ARROWCLOAK can protect the attention module. We will use $Q \cdot K^\top$ as an example to illustrate the protection process.

B.1 Limitation of OTP-based Encryption

Although OTP is utilized in Slalom to protect the input of the outsourced layers, it can not be used to protect the intermediate results of the attention module. The reason is that the decryption of OTP for the attention module needs to perform a matrix-matrix multiplication. The computation complexity of the multiplication is $O(n^2 l)$, which is the same as the original multiplication. TEE can directly perform the multiplication instead of decrypting the OTP.

Let the random mask be R_Q and R_K for Q and K respectively. The encrypted Q and K are computed as:

$$\tilde{Q} = Q + R_Q, \quad \tilde{K} = K + R_K.$$

The multiplication of $\tilde{Q}$ and $\tilde{K}$ is computed as:

$$\begin{aligned}\tilde{Q} \cdot \tilde{K}^\top &= (Q + R_Q) \cdot (K + R_K)^\top \\ &= Q \cdot K^\top + Q \cdot R_K^\top + R_Q \cdot K^\top + R_Q \cdot R_K^\top.\end{aligned}$$

TEE can recover $Q \cdot K^\top$ by computing

$$Q \cdot K^\top = \tilde{Q} \cdot \tilde{K}^\top - Q \cdot R_K^\top - R_Q \cdot K^\top - R_Q \cdot R_K^\top.$$

Note that, unlike the linear layer, $R_Q \cdot K$ and $Q \cdot R_K$ can not be precomputed in the offline phase. This is because K and Q are different for each input and must be computed online. The computation complexity of the multiplication is $O(nl^2)$, which is the same as $Q \cdot K^\top$. Therefore, the OTP-based encryption is not suitable for protecting the attention module.

B.2 ARROWCLOAK for Attention Module

ARROWCLOAK can directly apply to attention modules to protect the intermediate results. We first represent Q and K as the form of vectors:

$$Q = [\mathbf{q}^1, \ldots, \mathbf{q}^l]^\top, \quad K = [\mathbf{k}^1, \ldots, \mathbf{k}^l]^\top,$$

where $\mathbf{q}^i, \mathbf{k}^i \in \mathbb{R}^{1\times n}$ are the i-th row of Q and K respectively.

Let $\mathbf{v}_Q \in \mathbb{R}^{1\times n}$ and $\mathbf{v}_K \in \mathbb{R}^{1\times n}$ be the random vectors generated within the TEE. The obfuscation of Q and K is computed as:

$$\tilde{Q} = \Pi_1 \cdot [a_Q^i \cdot \mathbf{q}^i + b_Q^i \cdot \mathbf{v}_Q]^\top = \Pi_1 \cdot (D_1 \cdot Q + D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q),$$

$$\tilde{K}^\top = [a_K^i \cdot \mathbf{k}^{i\top} + b_K^i \cdot \mathbf{v}_K{}^\top] \cdot \Pi_2 = (K^\top \cdot D_3 + \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4) \cdot \Pi_2.$$

where $a_Q^i, b_Q^i, a_K^i, b_K^i \in \mathbb{R}$ are the obfuscation keys. The offloaded $\tilde{Q}$ and $\tilde{K^\top}$ multiplication can be represented as:

$$\begin{aligned}\tilde{Q} \cdot \tilde{K}^\top = \Pi_1 \cdot (&D_1 \cdot Q \cdot K^\top \cdot D_3 + D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q \cdot K^\top \cdot D_3 \\ &+ D_1 \cdot Q \cdot \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4 + D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q \cdot \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4) \cdot \Pi_2.\end{aligned}$$

Thus the deobfuscated $Q \cdot K^\top$ can be computed as:

$$Q \cdot K^\top = \Pi_1^{-1} \cdot D_1^{-1} \cdot (\tilde{Q} \cdot \tilde{K}^\top - OBS) \cdot D_3^{-1} \cdot \Pi_2^{-1},$$

where OBS can be represented as:

$$\begin{aligned}OBS = &D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q \cdot K^\top \cdot D_3 + D_1 \cdot Q \cdot \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4 \\ &+ D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q \cdot \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4.\end{aligned}$$

Note that the computation complexity of $D_2 \cdot \mathbf{1}_n^\top \cdot \mathbf{v}_Q \cdot K^\top \cdot D_3$ and $D_1 \cdot Q \cdot \mathbf{v}_K{}^\top \cdot \mathbf{1}_n \cdot D_4$ is $O(nl)$ because these operations are matrix-vector multiplication. Thus, the computation complexity to recover $Q \cdot K^\top$ is $O(nl)$, which is tolerable for the TEE.