GROUPCOVER: A Secure, Efficient and Scalable Inference Framework for On-device Model Protection based on TEEs

Zheng Zhang [1] **Na Wang** [1] **Ziqi Zhang** [2] **Yao Zhang** [3] **Tianyi Zhang** [1] **Jianwei Liu** [1] **Ye Wu** [3]

Abstract

Due to the high cost of training DNN models, how to protect the intellectual property of DNN models, especially when the models are deployed to users' devices, is becoming an important topic. One practical solution is to use Trusted Execution Environments (TEEs) and researchers have proposed various model obfuscation solutions to make full use of the high-security guarantee of TEEs and the high performance of collocated GPUs. In this paper, we first identify a common vulnerability, namely the fragility of randomness, that is shared by existing TEE-based model obfuscation solutions. This vulnerability benefits model-stealing attacks and allows the adversary to recover about 97% of the secret model. To improve the security of TEE-shielded DNN models, we further propose a new model obfuscation approach GROUPCOVER, which uses sufficient randomization and mutual covering obfuscation to protect model weights. Experimental results demonstrate that GROUPCOVER can achieve a comparable security level as the upper-bound (black-box protection), which is remarkably over $3\times$ compared with existing solutions. Besides, GROUPCOVER introduces 19% overhead and negligible accuracy loss compared to model unprotected scheme.

1. Introduction

With the evolution of deep learning techniques, machine learning as a service (MLaaS), especially DNN-based services, has become an important paradigm. Given that cloud-based MLaaS has large latency and high instability, manufacturers prefer to deploy model inference services to edge devices. These edge devices are usually equipped with high-performance accelerators such as GPUs, NPUs (Tan & Cao, 2022), etc. Deploying DNN to edge devices is particularly prevalent in applications like autonomous driving (Hu et al., 2019) and Internet of Things (IoT) monitoring (Hameed et al., 2022). However, putting model weights onto edge devices introduces a new threat to model owners: *the untrusted device owners may steal the intellectual property of DNN models* (Sun et al., 2021; 2023; Zhang et al., 2024).

Existing solutions to protect on-device DNN models can be categorized into two types: cryptographic solutions (Al-Rubaie & Chang, 2019) and Trusted Execution Environment (TEE)-based (Sagar & Keke, 2021) solutions. Both solutions can be augmented by GPUs. Among the two solutions, *the TEE-based approach is more practical for real-world applications* with high-performance requirements because it introduces acceptable computational overhead is less than $10\times$ (Tramer & Boneh, 2019) and no accuracy loss. On the contrary, cryptographic solutions introduce significant overhead (up to $1000\times$ (Ng & Chow, 2021)) and may cause an accuracy drop. In this paper, we aim to further improve TEE-based solutions.

Specifically, we focus on the **model obfuscation** technique that can utilize both the security of TEE and the high-performance of GPUs. The motivation for this technique is to securely offload the inference of computation-intensive layers (e.g. linear and convolutional layers) from TEE to GPUs. For each layer, given the computation operands (inputs and layer weights), TEE first obfuscates the operands and sends the obfuscated operands to the GPU. GPU computes the obfuscated layer output, sends the output back to TEE, and TEE recovers the authentic layer output. In this way, the untrusted world outside TEE (e.g. OS and GPU) can not directly access valuable model weights and the model secrets are protected. Existing obfuscation techniques include minimal split (Zhou et al., 2023; Hou et al., 2021), weight scale (Shen et al., 2022) and matrix permutation (Sun et al., 2023).

We analyze the security of existing model obfuscation techniques and find a common drawback: **the fragility of randomness**, which is reflected in the fixed obfuscation strategy and preserving partial statistical features. Due to this draw-

[1]Beihang University [2]Peking University [3]Douyin Vision Co., Ltd., Beijing, China. Work was done when Zheng Zhang was an intern at Douyin Vision Co., Ltd.. Correspondence to: Na Wang <nawang@buaa.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

back, the adversary can recover secret model weights from the obfuscated model. For insufficient randomness, the existing defenses can be divided into two types. The first type of defense only modifies a fixed small amount of model weights (Zhou et al., 2023; Hou et al., 2021). However, this fixed modification approach can result in an extreme distribution of weight, making it easy to identify abnormal model weights. The second type sets the convolution kernel as the atom obfuscation unit and employs lightweight randomization (e.g. shuffle and scale) to obfuscate models. Nevertheless, convolution kernels of both secret and public models have similar statistical distributions, making obfuscated weights leak partial secrets.

As proof of concept attack, we evaluate the security of existing model obfuscation solutions under a practical threat model. The adversary's goal is to use statistical analysis to disclose the model obfuscation mechanism and try to recover the secret model. Following prior work (Zhang et al., 2024), the adversary can access the public pre-trained models and datasets. We find that for prior model obfuscation schemes (Zhou et al., 2023; Hou et al., 2021; Sun et al., 2023; Shen et al., 2022), the adversary can recover an average of 97% of the secret models' functionality. The attack performance is similar to the no-shield baseline.

To improve the security of model obfuscation solutions, we further propose a new efficient defense scheme, GROUPCOVER. GROUPCOVER consists of two insights: *sufficient randomization strategy* and *mutual covering obfuscation*. Both of them aim to fully randomize the DNN's weight distribution so that the features between the original model weights and obfuscated weights are indistinguishable. GROUPCOVER first reversely clusters the convolution kernels and then selects a random linear combination of kernels as obfuscation noise. Compared with prior model obfuscation techniques, this approach introduces sufficient randomness to prevent unusual distribution patterns while introducing negligible overhead.

We evaluate the performance of GROUPCOVER with prior model obfuscation techniques under the same adversary that analyzes the statistical difference of model weights. Experimental results show that GROUPCOVER is $3\times$ more effective than other obfuscation schemes. GROUPCOVER achieves a similar defense level as the security upper bound (the black-box defense). As to the cost, compared with model IP unprotected scheme, GROUPCOVER only introduces less than 19% computational overhead and about 0.01% accuracy loss. Since preprocessing of GROUPCOVER is lightweight and involves only addition and subtraction operations, GROUPCOVER is scalable to large CNN models with different architectures.

The contributions of this paper are as follows:

- We reveal a key vulnerability, the fragility of randomness, in existing model obfuscation schemes. Under a practical threat model, this vulnerability allows the attacker to recover 97% of the obfuscated model weights by carefully analyzing the statistical distribution of model weights.
- To mitigate this security risk, we propose a new model obfuscation scheme, GROUPCOVER, that uses sufficient randomization and mutual covering obfuscation to protect model IP.
- Experimental results and theoretical analysis demonstrate that GROUPCOVER improves the defense effectiveness by $3\times$ to the same security level of upper-bound protection (black-box defense). Besides, GROUPCOVER only introduces less than 19% computation overhead and negligible accuracy loss.

2. Related Work

2.1. Secure GPU-Outsourcing Inference Scheme

Privacy-preserving model inference is a rapidly emerging area of research. Researchers aim to securely outsource resource-intensive operations to untrusted hardware accelerators, aligning with industrial practices in the real world. For example, (Ng & Chow, 2021) uses cryptographic methods to achieve secure inference on the CIFAR-100 dataset using VGG16 in 0.4 seconds, with an accuracy rate of 73%. (Yousefpour et al., 2021) uses differential privacy (DP) to protect input data's privacy. However, these approaches often result in a significant loss of accuracy and fail to balance security and efficiency. Similarly, the TEE-shield approach, represented by TEESlice (Zhang et al., 2024) which also re-trains models, provides full model and privacy protection, but at an inference speed that is 4-6 times slower than pure GPU implementations. Some hardware-assisted solutions (Chakraborty et al., 2020; Alam et al., 2022) achieve higher efficiency while protecting model IP, but they introduce more attack surfaces compared to TEE-based schemes,

Table 1. Comparison of Secure Outsourcing Inference Scheme. Acc., Eff., Sec. are acronym for accuracy, efficiency, and security.

Scheme	Method	Acc.	Eff.	Data Sec.	Model Sec.	Pre-cost
GForce	MPC	○	○	●	●	○
Opacus	DP	○	●	●	○	◐
TEESlice	TEE	◐	◐	●	●	◐
HPNN	HW	◐	●	○	◐	●
Ours	TEE	●	◐	●	●	●

thereby reducing their security protection. The performance of these work is presented in Table 1, where ● stands for best.

2.2. TEE-shield Outsourcing Inference Scheme

TEE-based secure inference solutions achieve a better balance between security and efficiency. TEE provides a secure enclave ensuring the trusted execution of remote programs. Various TEE products, proposed and upgraded by different hardware manufacturers, are now widely applied. In terms of inference efficiency, the substantial computational overhead within TEE demands GPU acceleration. However, as GPUs fall outside the Trusted Computing Base (TCB), the data transferred to GPU is untrustworthy. The key research objective is to reconcile this conflict by securely offloading data to GPU for intensive computation.

Slalom (Tramer & Boneh, 2019) is the first scheme to use GPU acceleration for secure TEE-based inference. It is observed that linear operations account for over 95% of the model inference process, therefore offloading linear computations to GPU in a secure manner can significantly speed up inference. In a basic linear operation of the form $y = Wx + b$, if the input to the GPU is transformed as $x' \rightarrow (x + r)$, the GPU output becomes $y' = Wx' + b = Wx + b + Wr = y + Wr$. Relying on the security properties of one-time padding, both the input and output of the GPU are masked. TEE only needs to compute the noise Wr offline to safely use GPU power for the most burdensome computations. Subsequently, Goten (Ng et al., 2021) and Darknight explored other homomorphic transformations, such as MPC, which enhance in either security or functionality. However, the above schemes ignore the privacy of model parameters, as models are still offloaded to GPU in plaintext.

Table 2. On-device inference scheme using TEE shield and GPU acceleration. Those in bold are the target of the attack.

Literature	Obfuscation Strategy
Slalom, ICLR 19 (Tramer & Boneh, 2019)	Model IP unprotected
Darknight, MICRO 21 (Mo et al., 2020)	Model IP unprotected
Magnitude, TDSC 22 (Hou et al., 2021)	Obfuscate the largest weights
NNSplitter, ICML 23 (Zhou et al., 2023)	Modify minimum weights by reinforcement learning
SOTER, ATC 22 (Shen et al., 2022)	Shuffle and scale weights
ShadowNet, S&P 23 (Sun et al., 2023)	Shuffle, scale and add limited noise

2.3. TEE-shield Schemes Focus on Model IP Protection

To protect the model IP, researchers have improved these frameworks by introducing model obfuscation techniques, which are highlighted in bold in Table 2. By obfuscating the model parameters offloaded to GPU, the inference accuracy using the obfuscated model is significantly reduced. The general idea of these approaches is to recover the correct output using recovery parameters within TEE for model inference. All of these schemes introduce adversaries executing black-box attacks to assess the protection of model parameters. Experiments with norm clipping and fine-tuning attacks are also alternatively included. However, they miss considering the security of the obfuscation strategy, lacking this critical consideration in their design.

3. Evaluation of Existing Obfuscate Scheme

3.1. Threat Model

Adversary's Goal. The sole goal of the adversary is to steal model IP. For model parameters obfuscated and loaded onto GPU, the adversary aims to reverse the obfuscation and train an surrogate model under limited budgets. This model, sharing the same architecture as the secret model, seeks to achieve similar performance on the same tasks as the secret model.

Adversary Capabilities. Firstly, similar to (Orekondy et al., 2019), the adversary is allowed to perform limited correct inferences which is around 50 queries per class. Then, the adversary, acting as a high-level administrator, can access all non-secure information, like GPU memory and PCIe communication (Markettos et al., 2019). They also have knowledge about the model's task, architecture, and public pre-trained parameters, enabling them to adaptively conduct attacks. Finally, like similar approaches Table 2, we do not encompass security challenges specific to TEEs, such as side-channel attacks.

3.2. Privacy Leakage Analysis

Based on our observation, we identify a common issue with existing obfuscation approach: the fragility of randomness, which leads to model privacy leakage. We will delve into it in the following section.

Schemes Modifying Minimal Weights. Schemes (Hou et al., 2021; Zhou et al., 2023) are based on the empirical understanding that in machine learning, larger weights have a more significant impact on model predictions. Magnitude following this principle, modifies the largest weights. It replaces the top 1% of weights with random numbers, and restores them within the TEE. NNSplitter comes with a similar idea, but it obfuscates a minimal set of crucial weights, about 0.002% of parameters, by employing reinforcement

learning. The detailed method is shown in Appendix A.

These schemes attain effectiveness and stealthiness with minimal modifications, but consistently show a post-obfuscation trait: an anomalous distribution of large weight parameters. Theoretically, this stems from the static, non-random of the obfuscation method, driving the network's obfuscated outcome toward the '*optimal*': predominantly converting original weights into the largest ones. In our replication of the NNSplitter across different settings, we observe a clustering of modified weights near $0.5max(w_i)$, as illustrated in the Figure 1. This distinct parameter anomaly enable the identification of aberrant weight behavior. Identifying weights within this anomalous range highlights a potential security risk in these schemes. By eliminating the anomalous intervals, an intermediate model very similar to the secret model can be obtained.

Inspiration 1. Fixed-strategy obfuscation results in distributional anomalies, making secret model recovery easier through strategy reversal. This indicates the necessity of employing **random strategy** in obfuscation.

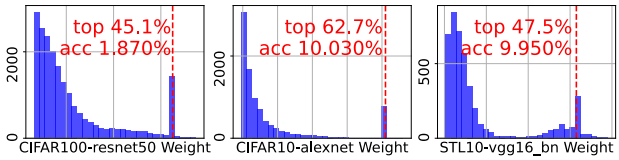

Figure 1. Weight distribution histogram of NNSplitter. The red line indicates $top = w/max(w)$ and the model accuracy acc.

Schemes to Completely Disrupt the Parameters. These schemes (Shen et al., 2022; Sun et al., 2023) focus on introducing minimal extra computation to thoroughly randomize model parameter distribution. Let's represent convolution filters as $W = [w_1, \cdots, w_n]$, where the convolution kernel w_i is the operational unit. Specifically, they seek reversible homomorphic transformations f^o such that $f^o(W)x = f^o(Wx) = f^o(y)$. Thus, correct inputs y can be recovered in TEE using f^{o-1}. The obfuscation methods of SOTER and ShadowNet are illustrated in Appendix B.

These schemes fundamentally adhere to the same encryption paradigm, employing scaling and shuffling techniques on each convolution kernel w_i. Note that the smallest operational unit is w_i, which shows the atomicity of convolution kernels. However, the effectiveness of scaling and shuffling in truly randomizing model parameter distributions is questionable. We select ImageNet (Deng et al., 2009) as the publicly available pretrained model, with the settings of the secret model displayed at the top of Figure 2. The results show a high cosine similarity distance between pretrained and secret models, indicating a near-perfect match in the shuffled convolution kernel indices across various networks. As depicted in Figure 2, the dimensionality-reduced (Van der Maaten & Hinton, 2008) weights of both networks are presented, with lines marking correct matches. Crucially, most layers allowed tracing back to the original indices before shuffling using publicly available pretrained weights.

Inspiration 2. Efficiency dictates convolution kernels as the atomic obfuscation units, while security necessitates disrupting each kernel's distribution. Therefore, **mutual covering** kernels rather than individually altering them optimally balances security and efficiency.

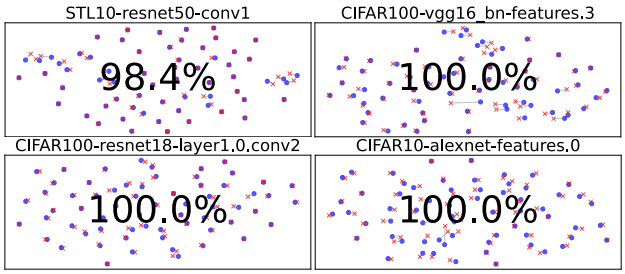

Figure 2. The similarity between convolution filters of obfuscated model and public model under t-SNE dimensionality reduction.

3.3. Attack Settings

To evaluate the security risks of prior schemes, we use three datasets STL10 (S10), CIFAR10 (C10), CIFAR100 (C100) and four model architectures AlexNet, ResNet18, ResNet50, VGG16), aligning with previous studies. We assume models trained on specific datasets as secret models. ImageNet is chosen for similar tasks as the pretrained model, which is accessible from internet as public models. We replicate obfuscation methods from schemes NNSplitter (Zhou et al., 2023), Magnitude (Hou et al., 2021), SOTER (Shen et al., 2022) and ShadowNet (Sun et al., 2023) to generate the obfuscated models locally. The detailed replication settings are shown in Appendix C.

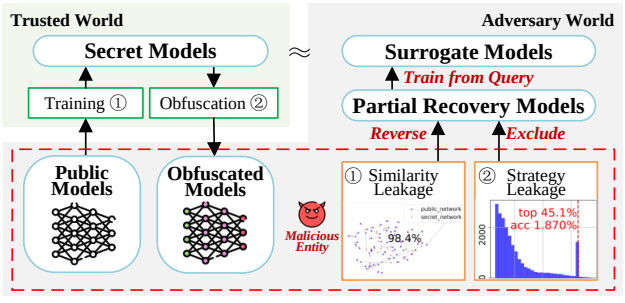

Figure 3. Attack pipeline.

Table 3. The attack results on NNSplitter, Magnitude, SOTER, and ShadowNet, are contrasted with two comparative baselines, No-shield and Black-box. The result $acc_1 \rightarrow acc_2$ indicates the accuracy improvement attributed to the attack.

		NNSplitter	Magnitude	SOTER	ShadowNet	No-shield	Black-box	Ours
AlexNet	C10	10.0% →81.4%	10.4%→70.3%	40.1%→74.2%	9.4%→58.1%	84.0%	17.1%	11.7%→17.3%
	C100	1.3%→56.0%	2.0%→46.5%	5.1%→51.5%	0.9%→42.6%	56.3%	21.2%	1.0%→24.4%
	S10	10.0%→75.5%	10.5%→55.4%	24.6%→55.1%	8.9%→59.4%	75.4%	32.1%	9.5%→43.6%
ResNet18	C10	10.0%→93.2%	10.5%→85.4%	37.5%→89.8%	11.1%→90.5%	94.5%	30.0%	10.7%→27.0%
	C100	1.0%→76.7%	0.9%→77.0%	6.0%→76.1%	0.9%→77.0%	79.6%	25.0%	1.1%→18.6%
	S10	10.0%→82.6%	10.4%→81.6%	18.5%→75.6%	9.0%→68.2%	83.1%	28.2%	9.2%→29.9%
ResNet50	C10	12.6%→91.2%	8.6%→85.4%	10.7%→89.8%	10.2%→90.9%	94.8%	24.9%	10.0%→26.5%
	C100	1.9%→77.5%	1.0%→80.4%	9.6%→78.6%	1.0%→77.6%	82.3%	17.9%	1.0%→20.8%
	S10	10.0%→85.2%	10.6%→85.0%	16.7%→78.2%	10.7%→78.4%	85.3%	26.0%	10.0%→30.3%
VGG16	C10	10.0%→90.8%	9.2%→86.2%	10.2%→87.9%	9.3%→89.1%	93.3%	39.8%	10.2%→31.2%
	C100	7.4%→68.9%	1.2%→69.1%	5.4%→69.3%	1.2%→70.0%	73.4%	18.9%	0.8%→24.0%
	S10	10.0%→90.5%	10.5%→89.2%	11.9%→82.2%	11.2%→84.2%	90.5%	32.7%	9.8%→28.4%
	Average	3.09×	2.91×	2.90×	2.82×	3.16×	1.00×	1.03×

3.4. Attack Pipeline

We construct a comprehensive attack pipeline to evaluate the security risks of current model obfuscation schemes to validate the theoretical analysis of their privacy leakages. Overall, our attack pipeline is depicted in Figure 3. We train the victim secret models on public pretrained models, using the obfuscated models as targets for the attack. The adversary first conducts adaptive statistical analysis to deduce the obfuscation method, then attempts to reverse the obfuscation. After roughly eliminating the randomness introduced by obfuscation, the adversary trains the intermediate model by queries. Following the design of (Orekondy et al., 2019), we allow the adversary an attack budget of 50 queries per class, using these 50 sample images and label results to retrain the partial recovery model. For specific datasets such as C100, the adversary has access to 5000 training images, which means we implies a considerable computational budget for the adversary.

The key step in our attack pipeline is the reverse obfuscation based on theoretical analysis. For NNSplitter and Magnitude, we locate the anomalous intervals using the statistical analysis. By adaptively adjusting the range of these intervals, we precisely identify the modified sections and replace their parameters with those from the publicly available pretrained model. For SOTER and ShadowNet, we use cosine similarity to find the correct convolution kernel indices. Based on the obtained indices, the convolution kernels are arranged to reverse shuffle operation in obfuscation appoach. Then we adaptively remeasure the size of the kernels. In fact, it's not necessary to restore the exact kernel size. Using the corresponding kernel values from a public model as a reference, we rescale the obfuscated weights to intervals of 0.1 in the range [0, 1], aligning them closely with the averages of the public model, thereby achieving high attack accuracy.

3.5. Result Analysis

The attack outcomes are detailed in Table 3. The performance against unprotected secret models, labeled as **No-shield**, simultaneously reflects their inference accuracy. For a baseline comparison, we integrate passive black-box attacks (Orekondy et al., 2019). The original obfuscated models' inference accuracy is close to random guessing. When introducing adversaries using our attack pipeline, the obfuscated models are partially restored, with final attack outcomes approaching the secret model's output. The final attack efficacy for the four types of schemes is 2.91×, 3.09×, 2.82×, and 2.9× that of the black-box baseline. Notably, for the minimally modifying NNSplitter scheme, few abnormal weights are removed, resulting in the highest attack quality at 97.8% of the no-shield accuracy. Conversely, ShadowNet, which completely disrupts kernel scale and arrangement, allow us to only roughly eliminate introduced randomness based on existing privacy leak aspects, resulting in a final attack effect at 91.8% of the original model.

Key observation. Pursuing optimal obfuscation creates significant statistical anomalies. Preserving convolution kernels' atomicity for efficiency allows reverse of obfuscation through similarity with publicly available model kernels. A thorough obfuscation approach must be designed to address these security risks.

4. Our Proposed Scheme: GROUPCOVER

Aligned with prior research, our scheme is crafted to achieve security, efficiency and scalability. These key aspects will define our paper's design objectives, guiding the detailed development of our approach.

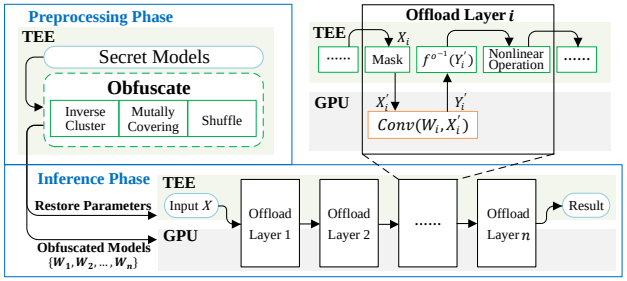

Figure 4. The process of GROUPCOVER.

4.1. Overall Process

The GROUPCOVER framework consists of two phases: offline preprocessing and online inference. As illustrated in Figure 4, the offline preprocessing takes secret model as input and randomly obfuscates its parameters without any input knowledge or fixed strategy, resulting in an obfuscated model whose inference accuracy is close to random guessing. The random numbers generated in this process are strictly confined within the TEE and protected by its security properties. During the online inference stage, the GPU performs computations on the obfuscated model, and the output is reconstructed in the TEE. This reconstruction relies entirely on the fixed random numbers from the preprocessing phase and the one-time padding generated for each prediction.

Algorithm 1 Inverse Clustering Vectors

Input: a layer of convolution kernels for secret model $V = [w_1, \cdots, w_n]$; the group size g.
Output: the inverse clusters: $group = [\{w_1^1, \cdots, w_g^1\}, \cdots, \{w_1^j, \cdots, w_g^j\}, \cdots, \{w_1^{n/g}, \cdots, w^{n/g}\}]$.
Init cosine similarity matrix csm of dimension (n, n).
while $g > 1$ **do**
  $g = g \,\%\, 2$
  for each w_i,w_j in V **do**
    Compute cosine similarity for w_i and w_j
    Assign the result to $csm[i, j]$
  end for
  Sort index by mean of csm columns to get $indices$
  Initialize $pairs$ and tmp lists
  for each i in $indices$ **do**
    Find j $s.t.$ minimize distance between $(V[i], V[j])$
    Concatenate$(group[i], group[j])$, add to tmp list
    Calculate mean$(V[i], V[j])$, add to $pairs$ list
    Set $csm[i, :], csm[j, :], csm[i, :], csm[:, j]$ to be inf
    Remove i, j in $indices$
  end for
  Update V with $pairs$ list and $group$ with tmp list
end while

4.2. Preprocessing Phase

Most linear operations can satisfy homomorphic properties within convolutional layers. We describe a convolutional layer as $Y = Conv(W, X)$, where Y and X represent the output and input of the layer. Meanwhile, $W = [w_1, \cdots, w_n]$ where n is the size of the output channel. For a multiplication and addition operation f^o, the transformed weights W satisfy $Conv(f^o(W), X) = f^o(Y) = Y'$, and in the TEE, using $f^{o^{-1}}(Y')$ restores the original output.

Followed the inspirations above, our goal is to disrupt the statistical distribution of atomic kernels cost-effectively, while confining operations to the convolution kernel level. The safest strategy would be random combination of all convolution kernels, equivalent to multiplying an $n \times n$ matrix with the weight W. However, this approach entails an $O(n^3)$ computational cost, impractical for efficiency-oriented inference schemes. Therefore, we propose a cost-effective obfuscation strategy that disguises individual atomic kernel features. We begin with a reverse clustering algorithm, grouping g vectors in W with the greatest cosine similarity distance, followed by inter-kernel covering within these clusters. This approach is intuitively motivated by the fact that two points with the greatest cosine similarity distance are easily distinguishable through statistical analysis. Therefore, clustering and obfuscating using points with the most disparate distributions can drive all kernels towards random distribution.

Our reverse clustering algorithm is detailed in Algorithm 1. Let the reverse clustering result for each convolutional layer be denoted as $group = [\{w_1^1, \cdots, w_g^1\}, \cdots, \{w_1^i, \cdots, w_g^i\}, \cdots, \{w_1^{n/g}, \cdots, w_g^{n/g}\}]$. Considering that output channels are usually aligned with powers of two and recognizing that smaller groupings facilitate more effective obfuscation, our scheme defaultly sets the grouping parameter $g = 4$. For each group $\{w_1^i, w_2^i, w_3^i, w_4^i\}$, we randomly select a transformation matrix composed of four linearly independent vectors $A = [\boldsymbol{a}^{(1)}, \cdots, \boldsymbol{a}^{(4)}]$ to randomly combine vectors within the cluster, where $\boldsymbol{a}^{(i)} = \{a_1^{(i)}, \cdots, a_4^{(i)}\}$. The obfuscated result for an individual convolution kernel is $w_j'^i = \boldsymbol{a}^{(j)}[w_1^i, w_2^i, w_3^i, w_4^i]^T = a_1^{(j)} w_1^i + a_2^{(j)} w_2^i + a_3^{(j)} w_3^i + a_4^{(j)} w_4^i$. Similarly, other kernels within the same group cover each other as $[w_1'^i, w_2'^i, w_3'^i, w_4'^i]^T = A \cdot [w_1^i, w_2^i, w_3^i, w_4^i]^T$. For a layer's weight matrix W, let $index(w_j^i)$ to be the index of w_j^i in W. We replace the vector of $index(w_j^i)$ with $w_j'^i$, and the final obfuscated model W' is the outcome of vector-by-vector replacement.

Within each cluster group, the convolution kernels mutually obfuscate each other's statistical feature. We employ a shuffle method, resulting in the obfuscated weight matrix $W' = P_\pi W'$, where P_π is an $n \times n$ matrix, with $P_\pi(i, j) = 1$ if $\pi(i) = j$, and 0 otherwise.

We visualize the preprocess process in Figure 5, depicting a t-SNE reduction (Van der Maaten & Hinton, 2008) of the first layer convolution kernels to a two-dimensional plane.

The red points represent the secret model, and the blue points represent the public pretrained model. For clarity, we focus on 16 kernels. The left subfigure shows grey lines connecting kernels of matching indices, demonstrating their similar distribution in both models. The farthest four kernels are chosen through reverse clustering, as shown in middle subfigure, where red lines indicate a cluster. After linear combination and shuffling, the original indices become unrecognisable, as shown in right subfigure. This disruption completely obscures the statistical features of the kernels.

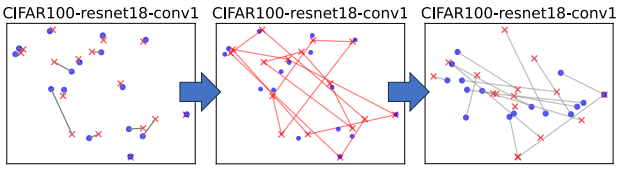

Figure 5. Visualisation of the pre-processing process.

Table 4. The program process of GROUPCOVER.

Data: input X, secret weight $W = [w_1, \cdots, w_n]$, output Y
Operation: random permute P_π, convolution $Conv(W, X)$
Preprocessing Phase:
Find inverse clusters V. For each group $[w_1, w_2, w_3, w_4] \in V$, $w'_i = \sum_1^4 a_j^i w_j$, where $\{\mathbf{a}^i\}$ are linearly independent.
Shuffle weight matrix. Rearrange W into $W' = P_\pi W$.
Generate noise offline. Compute $M = Conv(W', R)$.
Inference Phase:
Mask input $X' = X + R$. For each R, M is available.
Offload Conv in GPU. Sends X' to GPU, receive the result $Y' = Conv(W', X') = f^o(Y) + M$.
Restore output Y *in TEE.* Check the integrity and compute $Y = f^{o-1}(Y' - M)$. Use Y to generate the next input.

4.3. Inference Phase

The main objective of online inference is to accurately reconstruct outputs within the TEE. For this purpose, we divide the entire inference process into several offload layers, each bounded by a linear layer. The computationally intensive linear operations are securely offloaded to the GPU, while lighter non-linear and other operations are performed within the TEE. Here, we expand the flow of a single offload layer.

For our linear obfuscation method f^o, it satisfies $Y' = f^o(Y) = Conv(W', X)$. Specifically, for a cluster group of four output channels, the corresponding output results can be represented as Equation (1), where A is the restore parameters generated in preprocessing phase.

$$\begin{bmatrix} y'^i_1 \\ y'^i_2 \\ y'^i_3 \\ y'^i_4 \end{bmatrix} = P_\pi \begin{bmatrix} Conv(w^i_1, X) \\ Conv(w^i_2, X) \\ Conv(w^i_3, X) \\ Conv(w^i_4, X) \end{bmatrix} = P_\pi A \begin{bmatrix} y^i_1 \\ y^i_2 \\ y^i_3 \\ y^i_4 \end{bmatrix} \tag{1}$$

It's straightforward to deduce that $f^{o-1}([y'^i_1, y'^i_2, y'^i_3, y'^i_4]^T) = A^{-1}P_\pi^{-1}[y^i_1, y^i_2, y^i_3, y^i_4]^T$. Similarly, for outputs of batch size Y, $f^{o^{-1}}(Conv(W', X)) = Y$ remains valid. For each layer's input X, the TEE masks it as $X' = X + R$, where R is a one-time padding equal in size to X, and the noise $Conv(W', R)$ is precomputed offline. As GPU receives the obfuscated model W' and encrypted data X', it performs the computationally heavy linear operation $Y' = Conv(W', X') = Conv(W', X) + Conv(W', R)$. Upon receiving Y', the TEE first verifies its integrity using Freivalds' algorithm as described in (Tramer & Boneh, 2019). The TEE then removes the mask $Conv(W', R)$ and applies f^{o-1} to revert $Conv(W', X)$. This process ultimately restores the correct output Y within the TEE. In summary, the program process of GROUPCOVER, including both the preprocessing and inference phases, is presented in Table 4.

5. Analysis and Experiments

5.1. Security Analysis

The core security threats faced in GPU-accelerated TEE-based inference schemes are twofold, which are from obfuscation approach and offload data in each queries. For the former, we assess GROUPCOVER using the same attack outlined in Section 3, and use statistical analysis to illustrate why GROUPCOVER possesses superior security properties. For the latter, we illustrate the robustness of the random numbers our approach relies on.

Close to Black-box. With regard to the paramount aspect of security, we evaluated the protective capabilities of GROUPCOVER on the same attack pipeline. The attack results, presented in Table 3, show that our scheme's effectiveness against these attacks is around $3\times$ of other schemes, which is close to the black-box baseline. Furthermore, to thoroughly discuss the security resilience of GROUPCOVER and black-box defense schemes, we conducted attack experiments under varying budgets (10, 25, 50, 150, 200, 250, 300 per class). In the same CIFAR100 task, there were no significant differences in the performance of GROUPCOVER and black-box defense schemes across different networks, as shown as Figure 6. Utilizing the Wilcoxon signed-rank test (Wilcoxon, 1992), we find the confidence level of the difference statistic between the two types of schemes to be $p = 0.10$. Statistically, this implies that there is no significant difference between GROUPCOVER and black-box defense schemes.

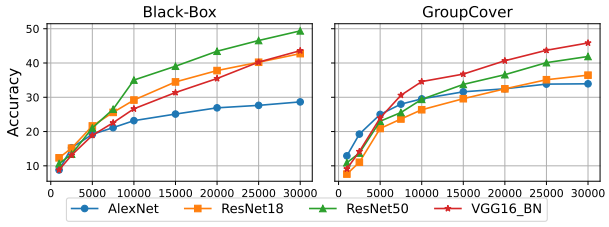

Figure 6. Attack defense performance under different budgets between Black-box scheme and GROUPCOVER.

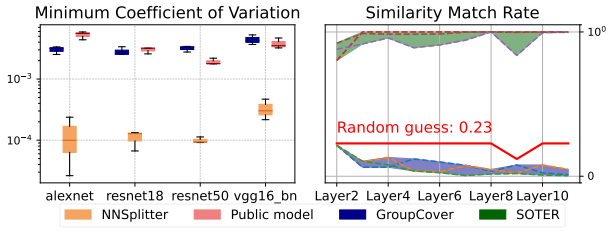

Figure 7. The minimum coefficient of variability in each small interval and similarity rate of convolution kernels between public models and obfuscated models.

No Anomalous Intervals Detected. In machine learning training, distribution density function of weights is generally smooth (Tsuchida et al., 2018). However, clipping methods such as those used in NNSplitter can produce discrete distributions within small intervals. To quantify the dispersion at each small interval, we use the coefficient of variability (CV) , which indicates the degree of data dispersion. Compared with the minimum CV results of our scheme and NNSplitter in the left of Figure 7, extremely low CV values indicate that data clusters at a fixed value. This empirical evidence suggests that NNSplitter exhibits dimensional disparities with public models, allowing identification and exclusion of abnormally intervals by norm clipping. In contrast, GROUPCOVER maintains consistency in the smallest interval values with public models, thus avoiding similar privacy leakage.

Low Similarity Rate. To verify that we have successfully obscured the statistical distribution of the atomic kernel, we use the same reverse methodology as in our attack pipeline to assess the security of the obfuscated model. Specifically, the convolution kernels are extracted from each layer of the obfuscated model. Their cosine similarity with corresponding parameters in the public model is computed to restore the indices prior to shuffling. For our scheme, the similarity match threshold is relaxed to include hits within the same cluster group. The right subfigure of Figure 7 shows the results from the first ten rounds under different network architectures. It is clear that SOTER's obfuscation results could be easily reversed due to the preservation of the atomic kernels' distribution. In contrast, our scheme effectively mask the distribution of atomic kernels by covering

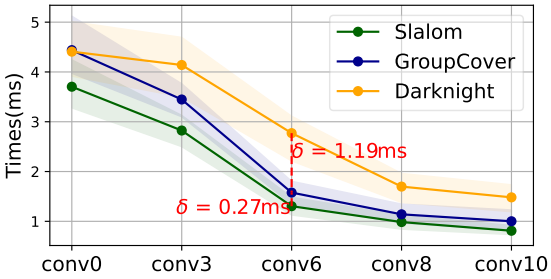

Figure 8. Inference time overhead for each layer under AlexNet. The solid line represents the cost of the representative scheme, while the shadows represent the distribution across all queries.

the most dissimilar convolution kernels. The success rate of adversaries using the public model to find the original groupings is lower than random guessing.

Completely Obscured by Random. During the inference process, data within the untrusted GPU, specifically W' and X' for each linear layer, is not TEE protected. We establish that statistical methods cannot reverse our obfuscation, which means GPU adversaries being limited to infer W from $Y = Conv(W, x)$ and X. For limited queries Q, we use a random quantity $Q \times \text{size}(X)$ in each layer for masking, equal to one-time padding, and its security is theoretically absolute with the seed stored in the secure TEE. This provides masking randomness for unlimited device queries that is equivalent to GPU-TEE communication, adheres to TEE encryption principles, and protects against GPU adversaries inferring model parameters.

5.2. Efficiency

The key challenge in the implementation was bridging communication between the TEE and the GPU, as the TEE theoretically does not trust the device. Our solution was to establish communication between the 8-core trusted VM and the RTX3070 through network interface, utilizing Torch's distributed RPC framework to manage the inference logic. We validate the efficiency of GROUPCOVER from two perspectives: performance in each layer and overall throughput, and details of configuration are given in Appendix D.

Offload Overhead. To illustrate the differences in efficiency overhead, we present the layer-by-layer offload costs under AlexNet in Figure 8. Since the inverse homomorphic operation of GROUPCOVER, like Slalom, is $O(n)$ in computational complexity, our performance overhead is largely consistent with Slalom. For example, in the middle convolutional layers, the overhead of GROUPCOVER differs from Slalom by only 0.27ms, while Darknight is 1.19ms higher. Overall, GROUPCOVER achieve model IP protection without introducing excessive overhead.

Table 5. Throughput(samples/sec) comparison and performance analysis. "No Sec." refers to execution entirely on the GPU.

Dataset	No Sec.	Slalom	GroupCover			
	Thp.	Thp.	Thp.	CPU	GPU	Transfer
C100	15118	5556	4465	0.47s	0.23s	1.54s
C10	15147	5670	4437	0.46s	0.23s	1.56s
S10	9241	4383	3699	0.56s	0.21s	1.39s
Average	2.53x	1x	0.81x	22.6%	10.1%	67.3%

Throughput. For confirmation that our conclusions are efficient, we perform throughput tests on different datasets and present the results in Table 5. In the same experimental environment, we test entirely GPU inference without TEE participation, the no-shield Slalom scheme, and our GROUPCOVER. In AlexNet model, our throughput achieves 32% of the pure GPU inference, with a loss of less than 19% compared to Slalom. This demonstrates that on edge devices, our scheme can fully exploit the capabilities of the accelerator, meeting real-time inference requirements while providing additional protection for model parameters.

Performance Analysis. We evaluated the inference performance on the entire test dataset, as shown in Table 5. Note that the transfer time includes only the communication time from the TEE to the GPU via the network interface. During inference, our GROUPCOVER incurs significantly more data transmission, resulting in substantial overheads that account for 67.3% of the execution time. Additionally, extra memory copying and data transfer introduce further computational overhead to the CPU. Employing a more efficient communication method between the TEE and GPU would substantially improve the throughput of GROUPCOVER.

5.3. Scalability

Our obfuscation scheme is designed for adaptive use within GPU-accelerated TEE-based inference frameworks. Maintaining prediction accuracy is crucial, so retraining or fine-tuning the secret model is generally unacceptable, with minimal loss in accuracy during online inference. Due to TEE's need for periodic destruction of programs and data, cost-effective local preprocessing is essential to generate new obfuscated models. Our implementation is also flexible, which indicates significant potential for GROUPCOVER.

Accuracy. By ensuring that each cluster's linear transformation matrix A in GROUPCOVER is invertible, the inference process theoretically incurs no loss in accuracy. However, numerical errors may arise in practical computations. We provide GROUPCOVER's accuracy performance across different datasets and network architectures in Table 6. The average loss is a mere 0.013%, indicating that GROUPCOVER maintains the initial performance of the secret model and is suitable for various fields where high accuracy is essential.

Table 6. The inference accuracy comparison.

Dataset	No-shield		GROUPCOVER		Average Loss
	AlexNet	VGG16	AlexNet	VGG16	
C100	56.31%	73.45%	56.23%	73.46%	-0.07%
C10	83.96%	93.29%	83.99%	93.27%	+0.01%
S10	75.36%	90.53%	75.34%	90.53%	-0.02%

Preprocessing Cost. To demonstrate GROUPCOVER's remarkably low preprocessing cost, we first compare it with other model IP protection schemes. It's important to note that leading approaches in this area typically require altering the network's structure or properties to achieve higher security or accuracy. For instance, cryptography and differential privacy-based methods necessitate model retraining. For vertical comparison, the TEESlice scheme requires retraining, while NNSplitter uses reinforcement learning for obfuscation, with an average time cost of 5 hours per device. In contrast to these intensive pre-processing steps, GROUPCOVER takes only a few seconds to obfuscate a model, showing a significant advantage in low pre-processing cost.

Implementation. Our RPC-based implementation enables GROUPCOVER to be applied in distributed scenarios. Enhancing communication between TEE and GPU, coupled with system-level optimizations, can substantially accelerate GROUPCOVER's inference efficiency. These improvements suggest that GROUPCOVER has the potential for large-scale secure inference applications.

6. Conclusion

We assess existing GPU-accelerated TEE-shield schemes for protecting model IP and identify privacy leakage issues in their model obfuscation methods. Exploiting these vulnerabilities, we design a statistical analysis-based model stealing attack, achieving $3\times$ of the accuracy compared to black-box baseline. This inspires to the security design of GROUPCOVER, focusing on sufficiently random obfuscation strategies and mutual covering of model parameters. Under similar attacks, GROUPCOVER achieves defense capabilities close to black-box schemes ($1.03\times$) while introducing moderate computational overhead (19%). Compared to similar schemes requiring model structural modifications, GROUPCOVER maintains inference accuracy and has minimal preprocessing time, offering the promising scalability.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Al-Rubaie, M. and Chang, J. M. Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy*, 17(2):49–58, 2019.

Alam, M., Saha, S., Mukhopadhyay, D., and Kundu, S. Nnlock: A lightweight authorization to prevent ip threats of deep learning models. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3):1–19, 2022.

Chakraborty, A., Mondai, A., and Srivastava, A. Hardware-assisted intellectual property protection of deep learning models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Dhanuskodi, G., Guha, S., Krishnan, V., Manjunatha, A., O'Connor, M., Nertney, R., and Rogers, P. Creating the first confidential gpus: The team at nvidia brings confidentiality and integrity to user code and data for accelerated computing. *Queue*, 21(4):68–93, 2023.

Hameed, S. S., Abdulshaheed, H. R., Ali, Z. L., and Gheni, H. M. Implement dnn technology by using wireless sensor network system based on iot applications. *Periodicals of Engineering and Natural Sciences*, 10(2):128–137, 2022.

Hou, J., Liu, H., Liu, Y., Wang, Y., Wan, P.-J., and Li, X.-Y. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19 (6):4270–4284, 2021.

Hu, C., Bao, W., Wang, D., and Liu, F. Dynamic adaptive dnn surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1423–1431. IEEE, 2019.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Markettos, A. T., Rothwell, C., Gutstein, B. F., Pearce, A., Neumann, P. G., Moore, S. W., and Watson, R. N. Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals. 2019.

Mo, F., Shamsabadi, A. S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A., and Haddadi, H. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pp. 161–174, 2020.

Ng, L. K. and Chow, S. S. GForce: GPU-friendly oblivious and rapid neural network inference. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2147–2164, 2021.

Ng, L. K., Chow, S. S., Woo, A. P., Wong, D. P., and Zhao, Y. Goten: GPU-outsourcing trusted execution of neural network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 14876–14883, 2021.

Orekondy, T., Schiele, B., and Fritz, M. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4954–4963, 2019.

Sagar, S. and Keke, C. Confidential machine learning on untrusted platforms: a survey. *Cybersecurity*, 4(1):1–19, 2021.

Shen, T., Qi, J., Jiang, J., Wang, X., Wen, S., Chen, X., Zhao, S., Wang, S., Chen, L., Luo, X., et al. SOTER: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 723–738, 2022.

Sun, Z., Sun, R., Lu, L., and Mislove, A. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1955–1972, 2021.

Sun, Z., Sun, R., Liu, C., Chowdhury, A. R., Lu, L., and Jha, S. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1596–1612. IEEE, 2023.

Tan, T. and Cao, G. Deep learning on mobile devices through neural processing units and edge computing. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pp. 1209–1218. IEEE, 2022.

Tramer, F. and Boneh, D. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019.

Tsuchida, R., Roosta, F., and Gallagher, M. Invariance of weight distributions in rectified mlps. In *International Conference on Machine Learning*, pp. 4995–5004. PMLR, 2018.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Wilcoxon, F. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pp. 196–202. Springer, 1992.

Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., et al. Opacus: User-friendly differential privacy library in pytorch. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

Zhang, Z., Gong, C., Cai, Y., Yuan, Y., Liu, B., Li, D., Guo, Y., and Chen, X. No privacy left outside: On the (in-)security of tee-shielded dnn partition for on-device ml. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 52–52, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

Zhou, T., Luo, Y., Ren, S., and Xu, X. NNSplitter: An active defense solution for DNN model via automated weight obfuscation. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 42614–42624. PMLR, 23–29 Jul 2023.

A. Detailed Obfuscation Methods of NNsplitter and Magnitude

NNSplitter (Zhou et al., 2023) employs a reinforcement learning approach to find an optimized obfuscation strategy. They define 'stealthiness' as making minimal modifications to the weights. To achieve this, they introduce an RL-based controller targeting the most crucial weights. With a loss function defined as $\mathcal{L}$ and the changes of weight as $\Delta\mathbf{W}$, their fixed optimization strategy can be described as:

$$\min_{\Delta\mathbf{W}'} \mathcal{L}(\Delta\mathbf{W}') = -\mathcal{L}_{\mathcal{M}}\left(f\left(\boldsymbol{x}; \mathbf{W} + \Delta\mathbf{W}'\right), \boldsymbol{y}\right) + \lambda \left\|\Delta\mathbf{W}'\right\|_0$$
$$\text{s.t.} \quad \alpha * \min\{w_i\} \leq w_i + \Delta w_i' \leq \beta * \max\{w_i\},$$

where f denotes the functionality of the DNN model $\mathcal{M}$, x is the training samples with y being the corresponding labels, and λ controls the sparsity of weight changes.

Depending on the network and hyper-parameters, the NNSplitter agent gradually reduces the number of convolution kernels to be modified. It alters parameters within the range of $[c \pm \epsilon]$, where c, ϵ is the hyper-parameters, to the interval $[\alpha \cdot min(w_i), \beta \cdot max(w_i)]$, thus minimizing the obfuscated model's accuracy.

Algorithm 2 Secure Versions Generation

Input: $W = \{W_1, \cdots, W_L\}; p.$
Output: W and $r = \{r_1, \cdots, r_L\}.$
for $l = 0; l < L; l++$ **do**
  $m \leftarrow \text{Mean}(W_l); s \leftarrow \text{STD}(W_l);$
  Let W_{top} be the top $|W_l| \cdot p$ weights of W_l;
  $r_l \leftarrow \emptyset;$
  for $w_i \in W_{\text{top}}$ **do**
    $r \leftarrow \text{Norm}(\text{mean} = m, \text{STD} = s);$
    $w_i \leftarrow w_i + (-w_i + r);$
    $r_l \leftarrow r_l \cup (-w_i + r);$
  end for
  Update W_l with W_{top};
end for

The Magnitude (Hou et al., 2021) obfuscation approach adopts a more straightforward method by selecting the top W weights with the largest magnitudes and substituting them with values drawn from a distribution identical to that of the secret model's parameters. The specifics of Magnitude's obfuscation algorithm are detailed in Algorithm 2.

B. Detailed Obfuscation Methods of SOTER and ShadowNet

SOTER (Shen et al., 2022) and ShadowNet (Sun et al., 2023) both protect the IP of model parameters through weight matrix modification. While they employ different architectures to articulate this process, they follow the same design rationale and cryptographic paradigm. Both treat linear layers as $W = [w_1, \cdots, w_n]$, where n represents the size of the output channel for convolutional layers and the first

dimension of the weight matrix for fully connected layers. Subsequently, they obfuscate these linear operations in comparable fashions. For instance, SOTER, focusing on scale, alters 20% of the parameters and then shuffles both plaintext and obfuscated weights before offloading to the GPU, ensuring the model distributed to the GPU remains non-functional. ShadowNet prioritizes security properties, scaling all weights w_i and disrupting the sequence of vectors in W. Despite introducing additional noise, for efficiency's concern, they expose this noise to the GPU, enabling noise elimination through simple differential attack. The program process of SOTER and ShadowNet is shown in Table 7.

Table 7. The program process of obfuscate and inference for SOTER and ShadowNet.

Data: input X, secret weight $W = [w_1, \cdots, w_n]$, output Y **Operation:** random permute P_π, convolute $Conv(W, X)$	
SOTER Obfuscation: obfuscate $\theta = 20\%$ weights $W' = f^o(W) = P_\pi(\mu w_i)$ **Offload Conv in GPU:** $Y' = Conv(W', X)$ $= f^o(Y) = P_\pi(\mu y_i)$ **Restore Output Y in TEE:** $Y = f^{o-1}(Y') = P_\pi^{-1}(\mu^{-1} y_i')$	**ShadowNet Obfuscation:** $w_i' \in W'$, $w_i' = \lambda_i w_i + f_{\lceil 0.2i \rceil}$ $W' = f^o(W) = P_\pi(\{w_i'\})$ **Offload Conv in GPU:** $Y' = Conv(W', X) = f^o(Y)$ $= P_\pi(\lambda_i y_i + f_{\lceil 0.2i \rceil} x_i)$ $R = Conv(\{f_i\}_{i \in [0.2n]}, X)$ **Restore Output Y in TEE:** $Y = f^{o-1}(Y')$ $= P_\pi^{-1}(\lambda_i^{-1} y_i' - R)$

C. The Replicate Details of Attacked Schemes

In configuring our hyperparameters for replication NNSplitter, we precisely define the range $[c - \epsilon, c + \epsilon]$ to ensure that the number of parameters modified within each layer do not exceed 0.002%. For the boundaries $[\alpha \cdot \min(w_i), \beta \cdot \max(w_i)]$, we select the values corresponding to the top 0.1% of the weights as the limits. We set the coefficients α and β to vary between 40% and 60%, thus providing a more concealed alteration than the original setup. This adjustment subtly affects the weight distribution, offering a stealthier approach to parameter modification while still adhering to the core principles of the NNSplitter method. On the other hand, Magnitude takes a broader range. It does not select weights based on their location within a range, but rather targets the upper echelon of weights by magnitude. Specifically, the top 1% of weights are replaced with values randomly generated around the aggregate mean of the parameters.

SOTER presents a middle ground with a prescribed obfuscation ratio of $\theta = 0.2$. This method does not discriminate between weights; instead, it randomly selects 20% of the weights for obfuscation. ShadowNet goes a step further by shuffling all convolution kernels. This comprehensive approach to obfuscation scrambles the order of the kernels. However, it reveals its mask weights $f_i, i \in \lceil rn \rceil$ to the GPU. This design decision means that the obfuscation parameter r, which we set to the default value of 0.2, does not impede the potential for reversing the obfuscation, as the random numbers can be easily eliminated through differential analysis. Follow the settings above, our replicate work is available in https://github.com/ZzzzMe/GroupCover.

D. Simulation Experiment Configuration

In response to the evolving landscape of Trusted Execution Environments (TEEs), which increasingly favor confidential virtual machine modes such as those introduced with Intel's TDX and the H100 series (Dhanuskodi et al., 2023), there is a marked shift toward ensuring security without necessitating changes to the upper-layer application code. Our research aligns with this trend by adapting our secure model obfuscation framework, GROUPCOVER, to operate within these next-generation TEEs. Leveraging the flexibility and wide adoption of the PyTorch framework, we have ensured that GROUPCOVER is compatible with established GPU-accelerated TEE inference frameworks such as Slalom.

To rigorously evaluate the overhead introduced by GROUPCOVER, we conducted an exhaustive analysis across a spectrum of model architectures, including, but not limited to, AlexNet, ResNet18, ResNet50, and VGG16_BN. This diverse set of models allows for a comprehensive understanding of the overhead implications across various architectural complexities and depths.

Our experimental design involved a consistent clustering size of 4, which was meticulously selected to maintain a balance between security and computational efficiency. The experiments are conducted over several datasets, including CIFAR10, CIFAR100, and STL10, with queries totaling 10k for CIFAR10, 10k for CIFAR100, and 8k for STL10, all processed under the same batch size of 128. This consistent approach across different datasets and models ensures the generalizability and applicability of our findings.