

# TensorShield: Safeguarding On-Device Inference by Shielding Critical DNN Tensors with TEE

Tong Sun<sup>1</sup>, Bowen Jiang<sup>1</sup>, Hailong Lin<sup>1</sup>, Borui Li<sup>2</sup>, Yixiao Teng<sup>1</sup>, Yi Gao<sup>1</sup>, and Wei Dong<sup>1</sup>

<sup>1</sup>The State Key Laboratory of Blockchain and Data Security

College of Computer Science, Zhejiang University, China

<sup>2</sup>School of Computer Science and Engineering, Southeast University, China

{tongsun,jiangbw,linhl,tengyixiao,gaoyi,dongw}@zju.edu.cn,libr@seu.edu.cn

## ABSTRACT

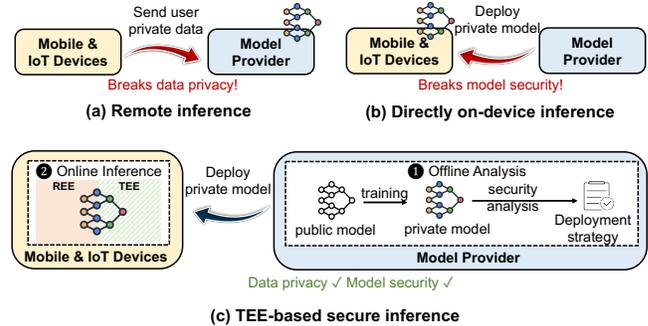
To safeguard user data privacy, on-device inference has emerged as a prominent paradigm on mobile and Internet of Things (IoT) devices. This paradigm involves deploying a model provided by a third party on local devices to perform inference tasks. However, it exposes the private model to two primary security threats: model stealing (MS) and membership inference attacks (MIA). To mitigate these risks, existing wisdom deploys models within Trusted Execution Environments (TEEs), which is a secure isolated execution space. Nonetheless, the constrained secure memory capacity in TEEs makes it challenging to achieve full model security with low inference latency.

This paper fills the gap with **TensorShield**, the first efficient on-device inference work that shields partial tensors of the model while still fully defending against MS and MIA. The key enabling techniques in TensorShield include: (i) a novel eXplainable AI (XAI) technique exploits the model’s attention transition to assess critical tensors and shields them in TEE to achieve secure inference, and (ii) two meticulous designs with critical feature identification and latency-aware placement to accelerate inference while maintaining security. Extensive evaluations show that TensorShield delivers almost the same security protection as shielding the entire model inside TEE, while being up to 25.35× (avg. 5.85×) faster than the state-of-the-art work, without accuracy loss.

## 1 INTRODUCTION

On-device inference has become an important paradigm for privacy- and latency-sensitive tasks on mobile and Internet of Things (IoT) devices [10, 17, 20, 27, 61, 70, 74]. Recent surveys [54, 67] have indicated that numerous mobile applications have implemented on-device deep learning models (DNNs) for a variety of uses, including liveness detection, video processing, and face recognition. The major advantages of on-device inference are obvious: (i) it protects user data privacy. As shown in Figure 1(a), remote inference requires transmitting user sensitive data to the untrusted clouds, compromising data privacy, (ii) it reduces the delays inherent in network communications, and (iii) it operates independently of internet connectivity.

However, on-device inference introduces new security threats to the deployed private deep neural network (DNN) models on billions of devices. As shown in Figure 1(b), attackers can easily obtain model weights and inference information (e.g., by dumping device memory), facilitating representative attacks, such as model stealing (MS) and membership inference attacks (MIA) at low costs.

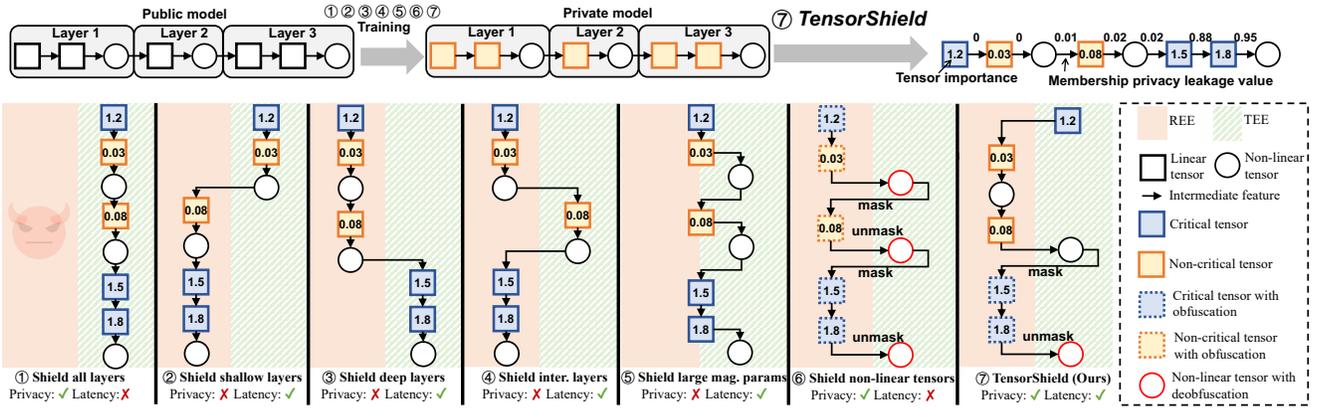


**Figure 1: Paradigms of interactions between user and private models. (a) Remote inference: users send data to the model provider. (b) Directly on-device inference: model providers deploy the private model in devices. (c) TEE-based secure on-device inference: model providers generate a secure deployment strategy (Ⓢ) and deploy the model with TEE (Ⓣ).**

In MS attacks, the adversary queries the target model with carefully crafted samples to maximally extract internal model information and then uses the returned query results to train a surrogate model [41, 49]. MS enables attackers to obtain a surrogate (a.k.a., shadow) model that closely replicates the functionality of the deployed model, leading to the leakage of private assets of the model provider. MIA allows attackers to query whether a sample is in the training set, resulting in sensitive data leakage [8, 28, 38]. These attacks could lead to severe security threats to numerous IoT applications. For example, attackers can launch MS attacks to obtain a surrogate model so as to construct adversarial samples, misleading the vision recognition systems of autonomous vehicles. Attackers can also launch MIA attacks to a model on health monitoring devices, causing privacy leakage of the user’s health conditions.

Recently, TEE-based secure inference approaches attract much research attention [16, 22, 34, 38, 48, 52, 53, 75, 76], primarily due to its high efficiency compared with previous security approaches such as Homomorphic Encryption (HE) [9, 30, 63]. In TEE-based secure inference approaches (Figure 1(c)), the model provider can train their private model from a large public model using the private dataset. The training process can be monitored for security analysis. Given the private model, different security approaches may then conduct different strategies to safeguard the inference process.

Some approaches achieve secure inference by shielding all layers. For example, Occlumency [34] (Ⓢ of Figure 2) shields all layers by executing the entire model in the TEE, using techniques such as on-demand loading of the weights and partitioned convolution calculation to address TEE’s memory limitation. ShadowNet [53] and GroupCover [76] (Ⓣ of Figure 2) also shield all layers by placing



**Figure 2: An illustration of previous work for model protection. The model has three layers with eight tensors. ① shields all (three) layers [34]. ② shields one shallow layer (Layer1) [16] and ③ shields one deep layer (Layer3) [38]. ④ shields one random intermediate layers (Layer2) [48]. ⑤ shields large magnitude weights of each layer [22]. ⑥ shields non-linear layers and obfuscates all linear tensors (indicated by rectangles with dashed lines) and all intermediate features [53, 76]. TensorShield (Ours) shields critical tensors (indicated by blue rectangles) and only masks privacy-related intermediate features.**

all non-linear layers in the TEE. The linear layers can be protected in REE with obfuscating by matrix transformation. These approaches can cause a high execution overhead because of on-demanding or (de)obfuscation.

To reduce the execution overhead of on-device inference, some other approaches only shield partial layers. For example, Serdab [16] (② of Figure 2) shields the shallow layers in the TEE to protect general information. DarknetZ [38] (③ of Figure 2) shields the deep layers in the TEE to protect classifier information. Unfortunately, these approaches cannot fully defend against MS and MIA attacks. The underlying reason lies in that they often fail to shield the most critical part with respect to model security. The essential of MS is to steal the *decision capabilities* of the model. For example, in DNN for image classification, the decision capability determines the category of an image. If these capabilities are well shielded, we can fully defend against MS and MIA [75]. However, the decision capabilities of a model do not necessarily depend on the shallow layers or deep layers. The *key idea* of TensorShield is to identify the most critical part of a model at the granularity of tensors, which offers a finer granularity than layers. By shielding these critical tensors (in TEE or in REE via obfuscation), we can achieve a high level of security without a large execution overhead.

How to identify the critical tensors? We rely on the recent progress in eXplainable AI (XAI) technique [24, 25] which leverages the contribution of weight and gradient changes to evaluate tensor importance. However, directly employing XAI incurs inaccuracy for model security evaluation because it only focuses on the contribution to the loss function and ignores the evaluation of decision-making ability (cf. §4.1). To this end, we propose a new metric called attention transition between the public pre-trained model and the victim model to accurately evaluate tensor criticality (cf. §4.1). After assessing its criticality, TensorShield trains a surrogate model on the validation dataset to simulate an attacker. We leverage the convergence speed of the surrogate model to select critical tensors in the TEE (cf. §4.2).

We have further proposed two novel techniques to reduce the inference latency. (i) instead of previous work that masks all transmitted features between the TEE and REE via one-time padding (OTP), TensorShield only masks privacy-related features determined by the JS-divergence distance to reduce latency overhead of masking (cf. §5), and (ii) unlike previous efforts that primarily leverage the REE GPU for inference, TensorShield conducts fine-grained modeling of hardware platform capabilities to minimize inference latency (cf. §6).

We evaluate the TensorShield with four well-deployed DNN models and four datasets on two hardware platforms (cf. §8). Compared to existing approaches that shield partial layers [16, 38], TensorShield reduces MS and MIA accuracy by up to 45.97% (avg. 28.54%) and 38.33% (avg. 23.15%), respectively, and reduces inference latency by up to 7.17 $\times$  (avg. 1.49 $\times$ ). In comparison to approaches that shield all layers [34, 76], TensorShield can reduce inference latency by up to 25.35 $\times$  (avg. 5.85 $\times$ ) and achieves almost the same level of protection.

We summarize our contributions as follows:

- We propose a novel XAI technique for critical tensor identification to defend against MS. We propose a metric named attention transition to more accurately evaluate the criticality of tensors.
- We propose a critical feature identification to defend against MIA. We leverage JS-divergence to assess membership privacy leakage and selective mask features via the OTP.
- We propose a latency-aware placement technique, the first to determine the placement of shielded tensors by jointly considering the hardware capabilities and the criticality of tensors and features.
- Our thorough evaluation shows that TensorShield offers almost the same security guarantee as shielding the entire DNN models inside TEE with over 25 $\times$  inference performance improvements than the state-of-the-art works.

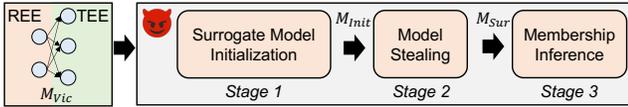


Figure 3: A three-stage attack pipeline.

## 2 BACKGROUND AND THREAT MODEL

### 2.1 Background

**Trusted Execution Environment.** TEE (e.g., ARM TrustZone [4]) provides a physical isolation scheme in the hardware devices that separates memory into the REE and TEE, where the REE can communicate with the TEE by invoking a secure monitor call. This setup ensures that only legitimate users can access the secure world, while attackers are blocked.

**On-device inference attacks.** On-device inference can fully preserve data in-situ, making it a widely utilized approach in privacy-sensitive applications [10, 17, 20, 27, 61, 70, 74]. However, deploying third-party private models on devices poses security risks [37, 39]. As shown in Figure 3, two prominent types of inference attacks that threaten model privacy are model stealing (MS) and membership inference attacks (MIA). First, the attacker infers the architecture of the victim model based on the REE part and the model output with existing techniques. Then, the attacker chooses a public model  $M_{Pub}$  (with the same architecture) as  $M_{Init}$ . Lastly, the attacker transports the model weights in the unshielded (i.e., REE) part of  $M_{Vic}$  to the corresponding parts of  $M_{Init}$ .

The goal of model stealing (a.k.a., model extraction) is to extract the parameters from a target (a.k.a., victim) model. The adversary queries the target model with carefully crafted samples to maximally extract internal model information and then uses the returned query results to generate an auxiliary dataset and train a surrogate model. Ideally, an adversary will be able to obtain a surrogate (a.k.a., shadow) model with very similar performance as the target model. More formally:

$$\text{MS: } M_{Vic}, \mathcal{D}_{aux} \rightarrow M_{Sur}$$

where  $M_{Vic}$  is the victim model,  $M_{Sur}$  is the surrogate model, and  $\mathcal{D}_{aux}$  is an auxiliary dataset.

MIAs pose serious privacy risks, leading to extensive research. These attacks aim to identify whether a particular sample was used in training a model. In black-box MIAs, attackers use model outputs and auxiliary data without accessing internal details. In contrast, white-box MIAs exploit both model outputs and internal information like gradients and activations to improve their effectiveness. More formally, given a target data sample  $x_{target}$ , and a stolen model  $M_{Sur}$ , and an auxiliary dataset  $\mathcal{D}_{aux}$ , an MIA can be defined as:

$$\text{MIA: } x_{target}, M_{Sur}, \mathcal{D}_{aux} \rightarrow \{\text{member, non-member}\}.$$

The term MIA accuracy refers to the accuracy of the surrogate model  $M_{Sur}$  for the member prediction task. The term MS accuracy refers to the accuracy of the surrogate model  $M_{Sur}$  for the victim’s original task. Thus, the lower these two accuracies, the more effective the protection is considered. In this paper, we use  $Acc_{MS}^{AllShield}$  and  $Acc_{MIA}^{AllShield}$  to represent the MS accuracy and MIA accuracy achieved by shielding the entire model with TEE, which represents the lower-bound level of security protection.

### 2.2 Threat Model

**Device platform.** In this paper, we focus on mobile and IoT devices requiring low-overhead protection approaches. We assume these devices are equipped with TEE, such as ARM TrustZone, which is prevalent in mobile and IoT devices [43]. Furthermore, we assume the TEE ensures the protection of data privacy and code execution against unauthorized access. Conversely, the REE is completely under the control of attackers. Side-channel attacks [6, 14, 36, 44, 71, 72] that could potentially lead to the leakage of sensitive information from the TEE are outside the scope of our study.

**Defender.** In our context, the defender is the model provider. Consistent with previous TEE-based secure inference methods, our goal to protect the deployed private model from unauthorized access and to prevent training dataset privacy breaches. Specifically, this paper aims to degrade white-box attacks to label-only black-box attacks, effectively achieving the protection level of enclosing the entire model within the TEE. We adopt the security assumption of a black-box baseline where the TEE fully shields the DNN model and returns only prediction labels, considered the upper-bound level of security protection offered by previous approaches. Following recent literature [75], we do not seek to entirely prevent information leakage from TEE outputs, such as prediction labels. To ensure high-quality ML services, the model provider ensures the accuracy of models. The system detects any fault injections in parameters or modifications in inference results that could compromise integrity. Advanced protective techniques safeguard the model before it is securely transmitted to edge TEEs using robust encryption and authentication methods. Once transmitted, the TEE selectively offloads the necessary model segments into the REE. We consider the transmission and offloading processes secure and tamper-proof [57]. As the model is deployed on the edge, protecting users’ private inputs is unnecessary.

**Adversary.** We assume the attacker possesses considerable power, enabling control over all aspects of the device environment excluding the TEE. We recognize that effective security mechanisms should not depend on the secrecy of the method itself, hence, the attacker is fully informed about the protection strategy implemented by the defender. We assume that both the model owner and the attacker can use the public model  $M_{Pub}$  on the Internet to improve the accuracy of the model or attacks, a realistic setting for modern on-device learning tasks. Consistent with previous work, we assume the private (i.e., victim) model is trained based on a pre-trained public model. The attacker can infer the architecture of the whole protected model, or an equivalent one, based on the public information (e.g., inference results or the unprotected model part). Besides, we assume that the attacker can query the victim model for limited times ( $\leq 1\%$  of the training data), a practical assumption shared by previous work [52, 75].

## 3 SYSTEM OVERVIEW

Figure 4 shows the workflow of TensorShield. For protection, we first need to identify the critical tensors (cf. §4) given the private model as well as the private dataset and the pre-trained public model. Each linear tensor (indicated by a rectangle) obtains a criticality value as shown in Figure 4(⊙). If a tensor has a large criticality value, it has a large contribution to the final model stealing accuracy.

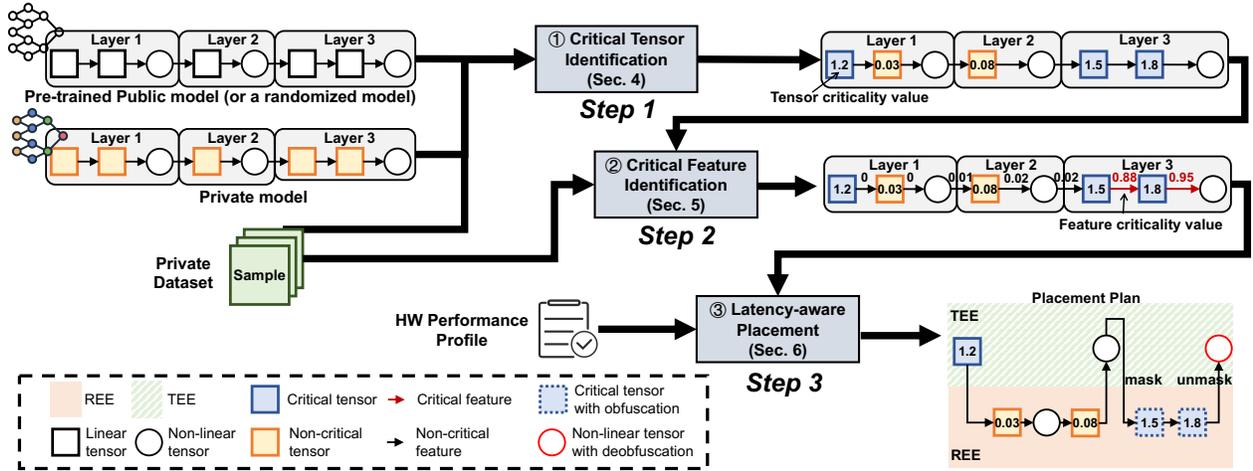


Figure 4: Workflow of TensorShield.

Note that non-linear tensors (indicated by circles) do not obtain criticality values. This is because they do not contain trainable parameters so they do not require protection. We then select a set of critical tensors (indicated by blue rectangles) for protection so that the model stealing accuracy does not exceed a given threshold (e.g.,  $Acc_{MS}^{AllShield} + 1\%$ ).

We next need to evaluate the criticality value of each cross-tensor transmission (i.e., the intermediate feature exchanged between two subsequent two tensors), as shown in Figure 4(2). We then select a set of critical intermediate features (cf. §5) so that the membership inference attack accuracy does not exceed a given threshold, e.g.,  $Acc_{MIA}^{AllShield} + 1\%$ . These intermediate features require masking and unmasking if the transmission across the TEE and REE boundary. No action will be taken for the intermediate features if the two subsequent tensors are placed at one side.

Finally, we need to decide how to place the tensors by jointly considering the criticality value of each tensor and intermediate features as well as the hardware performance profile (cf. §6), as shown in Figure 4(3). Each critical linear tensor can be placed in TEE or in REE via obfuscation, depending on the execution performance. The placement of the non-linear tensor depends on its previous linear tensor: if the previous linear tensor is placed in REE by obfuscation, the non-linear tensor should be placed in TEE because the protection via obfuscation only takes effect for linear tensors. If the transmission between two linear tensors is critical and the linear tensor is placed in REE, then its preceding and succeeding non-linear tensors should be placed in TEE for masking and unmasking. Transmission across the TEE and REE should be protected if it is identified as critical.

It is worth noting to emphasize some major differences compared to existing works: (1) Achieving the same level of protection, TensorShield protects far more less tensors. In our example, we only need to protect three linear tensors while shielding all layers approaches need to protect all eight tensors. (2) TensorShield selectively masks intermediate features related to membership privacy. For example, we only perform one pair of masking and unmasking operations while existing works require two pairs of masking and

unmasking operations for all intermediate features. (3) The execution of critical tensors can be either in TEE or REE via obfuscation depending on the execution efficiency. For example, the first tensor is placed in TEE while the sixth and seventh tensors are placed in REE via obfuscation. Note that mobile GPU has limited computation capability, rendering execution in TEE without obfuscation more computationally efficient.

## 4 XAI-BASED CRITICAL TENSOR IDENTIFICATION

### 4.1 Criticality Value Evaluation

As we have described in §3, we have to identify a set of critical tensors to be protected, i.e., either in TEE or in REE via obfuscation. We say that a set of tensors to be critical if we can use the corresponding parameters and structures to train a surrogate model which leads to a high model stealing accuracy. Consider a victim model’s accuracy is 90%. Protecting the entire model may lead to an accuracy of 60%. A surrogate model may achieve a high accuracy of 85%.

Given a set of protected tensors, we can measure the model stealing attack accuracy by simulating the attack. The simulation involves the following steps: (1) Randomly selecting a small sample (less than 1% of the training set) from the victim model’s validation dataset and erasing labels, then inputting it to the victim model to generate a pseudo-labeled dataset. (2) Using a pre-trained public model with the same structure as the victim model as the surrogate model, and initializing the unshielded victim model’s tensors to the corresponding surrogate model. (3) Training the surrogate model on the pseudo-labeled dataset (typically for 100 epochs). (4) Testing the surrogate model accuracy with the validation dataset as the model stealing accuracy.

However, identifying the set of critical tensors requires testing all possible combinations of tensors, which is always infeasible since well-deployed DNN models usually contain a large number of tensors.

We would like to exploit the recent advances in eXplainable AI (XAI) [24, 25] to give each tensor a criticality value which correlates the corresponding tensor to the final model stealing accuracy.

Intuitively, if a tensor has a large criticality value, it has a large contribution to the final model stealing accuracy.

Existing efforts [25, 73] suggest that tensor influences are not equally significant. Specifically, some tensors are considered “ambient” when a model shows negligible performance degradation after reinitializing or randomizing them. In contrast, tensors that cause significant performance degradation when altered are termed “important”. In other words, certain tensors are deemed “important” because they have a substantial impact on the decision boundary, while others are considered “ambient” due to their smaller effect. ElasticTrainer [25] measures the importance of tensors by evaluating the cumulative gradient changes of their weight updates during training. This metric aggregates how each weight update contributes to the reduction of training loss, and its computation naturally incorporates the impact of weight dependencies in training.

Directly applying this approach incurs two significant problems. (1) Overemphasis on tensors which can be easily trained by the attacker. This is because contribution to the reduction of training loss does not necessarily mean contribution to the model stealing accuracy. An important tensor may not necessarily be worth protecting since it can be easily trained by the attacker and its leakage does not lead to a significant increase in model stealing accuracy. In particular, we often find that ElasticTrainer overemphasizes tensors with a large number of parameters. (2) Overemphasis on tensors which be easily acquired from the pre-trained public model.

To address these problems, we introduce a novel criticalness metric for a DNN tensor  $k$  in a specific training epoch as:

$$I_k = \underbrace{\sum_i \frac{\partial L}{n \partial w_i^k} \Delta w_i^k}_{\text{Intrinsic tensor importance}} \times \underbrace{\left(1 - \cos(f(M_{Vic}^k), f(M_{Pub}^k))\right)}_{\text{Attention transition}}, \quad (1)$$

where  $L$  denotes the training loss function,  $w_i^k$  denotes the  $i$ -th weight in tensor  $k$ ,  $n$  is the number of weights in tensor  $k$ ,  $\Delta w_i^k$  is the recent update of  $w_i^k$  in the training epoch,  $\cos(\cdot)$  is the cosine similarity function, and  $f(\cdot)$  is the evaluation function of Grad-Cam [47]. We show the explanations in the following.

There are two terms in Eq. (1):

- The first term, intrinsic tensor importance, quantifies the contribution of each weight update to the reduction of training loss, utilizing a gradient computation that mirrors the backward pass, thereby naturally accounting for weight dependencies during training. Compared to previous work [25], we introduce a new consideration: a regularization term that accounts for the average contribution of parameters within tensors. Recent neural network theory [51] reveals that in the over-parameterized regime, models become highly reproducible, with wide model architectures producing nearly identical decision-making capacities across training runs. By normalizing the impact of parameters according to their average influence, we address a limitation of previous methods that often overemphasize the importance of parameters solely based on their gradient magnitudes, without considering the collective behavior of parameters within large tensors. This regularization provides a more balanced assessment, preventing the overvaluation of tensors with large parameter numbers.

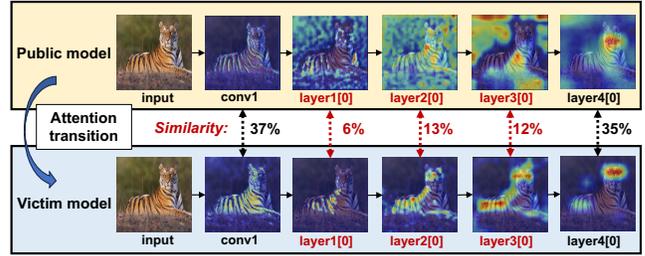


Figure 5: An instance of attention transition. Heat maps represent the importance of classifying model decisions.

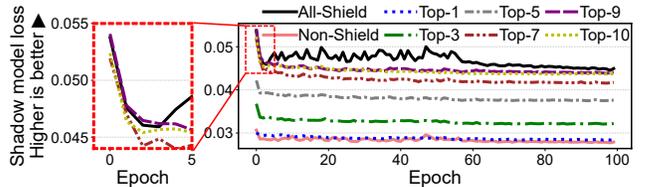


Figure 6: Model stealing loss values under different top-k strategies. Top-9 protects fewer parameters than Top-10 but also achieves the all-shield protection accuracy.

- The second term, attention transition, quantifies the tensors which can be easily acquired from the pre-trained model. We introduce attention transition into the tensor importance identification. Our key insight is that if a tensor in the victim model is intrinsically important, but similarly important in the public model, it may not need protection. Specifically, if the public model inherently possesses the capability for a task (e.g., classification decisions), an attacker could learn this capability without needing to steal parameters from the victim model. Figure 5 shows an example of attention transition using the ResNet18 victim model trained with CIFAR100 dataset and a pre-trained public model. We observe that the attention patterns of the tensor layer1.conv0, layer2.conv0, and layer3.conv0 are dissimilar between the victim model and the public pre-trained model, indicating that the victim model has learned new representations during training.

## 4.2 Critical Tensor Selection

Selecting critical tensors to achieve a pre-defined MS accuracy threshold in a cost-effective selection method is a challenge. To pinpoint the top-k tensors for protection, we utilize the validation data from the public dataset to simulate the MS attack. However, selecting from top-1 to top-k tensors (where  $k \leq \#$  of tensors) proves time-intensive given the extensive tensor numbers in modern DNNs. Once  $k$  is set, evaluating the theft precision typically requires extensive training (e.g., 100 epochs [75]).

To design our tensor selection mechanism, we make a fundamental *observation* that is unique in model stealing where the convergence speed of the shadow model’s loss function directly reflects the ultimate accuracy of the model stealing. We illustrate our approach using the VGG16\_BN architecture for both victim and shadow models, leveraging the CIFAR-10 dataset. Tensor importance in the victim model is quantified using Equ. 1. We protect the top-k layers while using the remaining layers to initialize the shadow model, with attack accuracy depicted in Figure 6. Our results highlight:

(1) Initial loss values provide a coarse measure of protection effectiveness, with initial epochs for top-3 and top-5 layers showing significantly lower values than the All-Shield scenario. (2) The rate of loss reduction over the first five epochs offers a fine-grained indicator of protection precision. Specifically, the rate for top-9 and top-10 closely mirrors that of All-Shield, while top-7 exhibits a more gradual decline.

Based on these findings, we propose a critical tensor selection method based on convergence speed. The key insight is that instead of waiting for the shadow model to converge, we can assess the accuracy of the top-k shadow model by comparing its convergence speed and initial loss function values with those of the model under All-Shield conditions. We initiate by evaluating loss variations across  $M$  iterations under non-shield and all-shield conditions. For each  $k$ , initial loss values for the corresponding shadow model and the rate of loss reduction over the first  $m$  epochs (e.g.,  $m=20$ ) are computed. If the protection accuracy of the all-shield scenario is met, we methodically decrease the percentage of protected tensors in each layer of the top-k. This process is repeated until the protection precision aligns with the threshold of the all-shield scenario.

## 5 CRITICAL FEATURE IDENTIFICATION

After identifying a set of critical tensors for protection to defend against MS, we have to identify critical intermediate features in order to defend against MIA.

If these intermediate features can be directly observed by the attacker, they can be used to train a membership attack binary predictor (i.e., predicts whether or not the input sample is in the training dataset). We say that the intermediate features are critical if their use leads to a high membership inference accuracy. Note that the lower bound of MIA accuracy is 50%, i.e., attackers can only make random guesses without additional membership privacy information.

The key to preventing privacy leakage of intermediate features is to perform masking so that these features cannot be observed when they are transmitted from TEE to REE. For example, existing work applies one-time padding (OTP) to mask all intermediate features transmitted from the TEE to the REE. The features are then unmasked in the TEE after being transmitted back from the REE to the TEE to recover the original values.

However, masking all intermediate features imposes significant execution overheads, especially on mobile and IoT devices. For example, ShadowNet incurs 160% and 150% execution time overheads on the Hikey960 platform for MobileNet and ResNet, respectively, due to the need to refresh masks [53].

Like identifying critical tensors, we would also like to assign each intermediate feature a criticality value to correlate them to the final MIA accuracy. If the intermediate features receive a higher criticality value, they have a larger contribution to the MIA accuracy. After the assignment of criticality values, we can select the top-k intermediate features for masking until we have achieved a predefined threshold of MIA accuracy, e.g., a slight increase compared with masking all intermediate features.

The key insight of our approach is to measure differences the intermediate features exhibit when the target model is given member samples and non-member samples. This is because the greater the

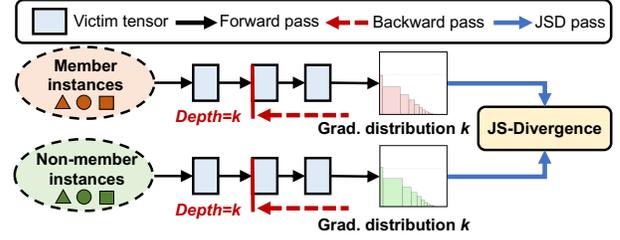


Figure 7: Compare the gradient and feature distributions of member and non-member inputs through JS-divergence.

distinction in the internal behavior of the model between member samples (i.e., samples in the victim’s training dataset) and non-member samples (i.e., samples not in the victim’s training dataset), the larger the membership privacy leakage, leading to a higher MIA accuracy.

We propose a novel membership privacy criticality metric for an intermediate feature, which evaluates membership privacy by calculating the JS-divergence distance of internal information (e.g., gradients) distributions for both members and non-members as:

$$JSD(p(z)||q(x)) = \frac{1}{2}KL\left(p\left\|\frac{p+q}{2}\right.\right) + \frac{1}{2}KL\left(q\left\|\frac{p+q}{2}\right.\right), \quad (2)$$

where  $G_p(x)$  and  $G_q(x)$  are gradient (and activation) distributions of member instances and non-member instances, respectively,  $KL(\cdot)$  is the Kullback-Leibler(KL) divergence function, and  $JSD$  is  $\in [0, 1]$ . If a pair of distributions is similar, the JSD is near 0; If a pair of distributions is dissimilar, the JSD is near 1. We select JS-divergence as the distance metric due to its symmetric properties, i.e.,  $JSD(p(z)||q(x)) = JSD(q(x)||p(z))$ . However, the widely used KL-divergence has an asymmetric nature, which does not uniformly characterize the distribution differences between non-member and member samples.

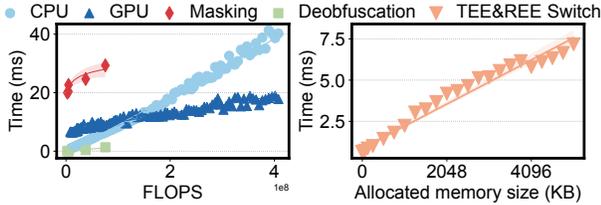
As shown in Figure 7, we assess the membership information exposure of various tensors in the victim model by applying Eq. (2) to these intermediate features.

Once the criticality values of all intermediate features are obtained, we can rank them from highest to lowest. A higher score indicates that an intermediate feature is more likely to leak membership privacy. We then select a set of critical features for protection so that the membership inference accuracy does not exceed a given threshold, e.g.,  $Acc_{MIA}^{AllShield} + 1\%$ . In our experimental observations, features with a criticality value of less than 0.1 typically do not affect membership inference attack (MIA) accuracy. Therefore, we can set a threshold to filter out features that have negligible distribution distinctions before performing MIA, thereby accelerating the evaluation process.

## 6 LATENCY-AWARE PLACEMENT

After selecting the critical tensors and intermediate features, TensorShield needs to decide the tensor’s placement based on the hardware performance profile.

Due to the unavailability of GPU acceleration within TEE, existing methods [53, 76] securely outsource the computation of the model’s linear tensor from the TEE to the REE via obfuscation. The obfuscated layers can then leverage GPU resources in the REE for accelerated inference. This approach is effective on powerful



**Figure 8: Left: Execution time w.r.t. different FLOPS. Right: TEE and REE switching time w.r.t. memory allocation size.**

platforms (e.g., clouds or PCs), where there is a significant disparity in computing power between CPUs and GPUs. However, our insight is that on mobile and IoT devices, heterogeneous processors exhibit varying affinities for different tensor computations. We illustrate the execution time for secure operations and various tensors on both CPU and GPU in Figure 8. The tests are performed on a Hikey960 mobile device. We use FLOPS to measure the computational cost of each tensor. In Figure 8, we can observe that existing methods overlook two key aspects: (1) Different FLOPS computations are better suited for different accelerators. For example, smaller FLOPS computations are more efficiently executed on CPUs, while larger FLOPS computations benefit from parallel execution on GPUs. (2) As the communication memory between the REE and TEE increases, the overhead associated with switching increases. This drives our effort to comprehensively establish a latency-aware placement method to minimize the inference latency while maintaining model security.

To address the placement issue, we establish a secure inference latency model and formulate it as a numerical optimization problem. We have introduced the basic placement rules in §3. Each tensor is either critical (i.e., requiring shielding) or non-critical. Critical tensors can be shielded in two ways: executed directly in the TEE or executed in the REE (after obfuscated offline but still need to be deobfuscated in the TEE online). To minimize the Trusted Computing Base (TCB) [19] size, non-critical tensors are executed directly in the REE. We consider there are two types of heterogeneous processors, i.e., CPU and GPU, while our formulation could be easily extended to more processors. There are three options for tensor execution: (1) in the TEE using the CPU; (2) in the REE using the GPU, and (3) in the REE using the CPU. We use a binary indicator  $x_{i,j}$  to denote whether the  $i$ -th tensor selects the  $j$ -th execution option in the following:

$$x_{i,j} = \begin{cases} 1 & \text{tensor } i \text{ is executed with the } j \text{ option} \\ 0 & \text{tensor } i \text{ is not executed with the } j \text{ option} \end{cases} \quad (3)$$

We further profile the execution time  $t_{i,j}$  of each tensor as follows:

$$t_{i,j} = \begin{cases} t_i^{(REE,CPU)} & j = 0 \\ t_i^{(REE,GPU)} & j = 1 \\ t_i^{(REE,CPU)} + t_i^{deobf} + t_i^{mask} & j = 2, \\ t_i^{(REE,GPU)} + t_i^{deobf} + t_i^{mask} & j = 3 \\ t_i^{(TEE,CPU)} & j = 4 \end{cases} \quad (4)$$

where  $t_i^{deobf}$  and  $t_i^{mask}$  denote the deobfuscation time for tensors and masking (including unmasking) for privacy-related features, respectively. Our optimization objective is to minimize the total inference time, which is determined by the computation time of

**Table 1: Device specifications in our evaluation.**

Device Name	CPUs	GPU	RAM	TEE RAM	REE OS	TEE OS
Hikey960 [2]	4xCortex-A73 (@2.36 GHz) 4xCortex-A53 (@1.84 GHz)	ARM Mali G71 MP8	4 GB	64 MB	Android 7	OP-TEE v3.4.0
Raspberry Pi 3B+ (RPi3B+) [3]	4xCortex-A53 (@1.40 GHz)	VideoCore 4	1 GB	32 MB	Raspbian OS	OP-TEE v3.4.0

each tensor and TEE and REE switching time:

$$\begin{aligned} \arg \min_x \quad & \underbrace{\sum_{i \in N, j \in \{0,1,2\}} x_{i,j} \cdot t_{i,j}}_{\text{computation time}} + \underbrace{\left( \sum_{i \in N-1} x_{(i-1),0} \oplus \left( \sum_{j=1}^4 x_{i,j} \right) \right)}_{\text{switch time}} \cdot t^{switch} \\ \text{s.t. (Correctness)} \quad & \sum_{j \in \{0,1,2\}} x_{i,j} = 1, \forall i \in N \\ \text{(Memory)} \quad & x_{i,0} \cdot m_i \leq M, \forall i \in N \end{aligned} \quad (5)$$

where  $t^{switch}$  denotes the switch time,  $m_i$  denotes the allocated TEE memory of the  $i$ th tensor, and  $M$  denotes a specified TEE available memory. Note that a switch in execution environments occurs only if the current tensor and the preceding tensor are processed in different execution environments. To model this, we use the  $\oplus$  operation to represent the switch.

Our problem is related to the traveling salesman problem and is NP-hard [65]. Fortunately, we have observed that the proportion of time spent on switching environments is trivial compared to the execution time, allowing us to disregard the overhead associated with environment switching. Consequently, by relaxing the condition, we can decompose the problem into multiple subproblems, enabling us to derive an approximately optimal solution with mature solvers:

$$x_{i,j} = \begin{cases} 1 & j = j_i^{opt} = \begin{cases} \arg \min_{j \in \{0,1\}} \{t_{i,j}\} & m_i \leq M \\ \arg \min_{j \in \{1,2\}} \{t_{i,j}\} & m_i > M \end{cases} \\ 0 & j \neq j_i^{opt} \end{cases} \quad (6)$$

## 7 IMPLEMENTATION

We implement the TensorShield as an end-to-end system consisting of two parts: offline profiling and on-device inference. In offline profiling, we leverage the Knockoff Net [41] as an emulator for MS to identify critical tensors. It is a standard query-based stealing technique where the attacker trains a model from a set of collected data labeled by the  $M_{Vic}$  [26, 49]. For MIA, we employ a transfer attack that builds  $M_{Sur}$  to imitate the behavior of  $M_{Vic}$  and infer the privacy of  $M_{Vic}$  from white-box information of  $M_{Sur}$  [45]. We leverage the attack implementation from a recent benchmark suite, ML-DOCTOR [37, 49]. We conduct victim and surrogate model training using PyTorch 2.1. To support on-device secure inference, we adopt DarkneTZ [38], the state-of-the-art on-device secure inference framework. We add  $\sim 2.4$ K LOC in C to DarkneTZ’s TEE modules and  $\sim 3.2$ K LOC in C to DarkneTZ’s REE modules for deobfuscating, masking, and unmasking in the TEE. Our code is open-sourced for public access<sup>1</sup>.

## 8 EVALUATION

The key takeaways of our evaluation are:

<sup>1</sup><https://github.com/suntong30/TensorShield>

**Table 2: Accuracy of  $\mathcal{M}_{vic}$  used in the evaluation.**

	CIFAR-10 [31]	CIFAR-100 [31]	STL-10 [11]	Tiny-ImageNet [32]
ResNet18 [21]	86.71%	60.72%	86.73%	42.96%
MobileNetV2 [46]	83.17%	50.06%	74.36%	41.26%
VGG16_BN [50]	85.28%	71.59%	80.20%	41.28%
ResNet50 [21]	86.47%	63.95%	87.28%	54.45%

- Across four datasets and four models, TensorShield can reduce inference latency and energy consumption by up to 25.35 $\times$  (avg. 5.85 $\times$ ) and up to 91.35% (avg. 58.66%) compared to the state-of-the-art obfuscation baseline [76], and up to 16.89 $\times$  (avg. 4.32 $\times$ ) and up to 98.35% (avg. 69.86%) compared to the state-of-the-art shielding the whole models baseline [34], while maintaining almost same MS (1.03 $\times$ ) and MIA (1.00 $\times$ ) accuracy.
- TensorShield’s critical tensor evaluator can reduce selected parameters by average relatively 70.32% (absolutely 17.62%) compared to the state-of-the-art XAI-based evaluating method [25] for achieving the same black-box MS and MIA accuracy.
- TensorShield’s improvement is significant across various workloads and platforms, including different input sizes, model architectures, and hardware devices.
- TensorShield’s system overhead is negligible.

## 8.1 Experimental Setup

**Platforms.** We conduct the evaluations on two types of mobile and IoT devices with different hardware specifications, all equipped with Armv8-A CPUs and ARM TrustZone. The detailed specifications are listed in Table 1. Hikey960 is a mobile development board equipped with HiSilicon Kirin960 SoC, and Raspberry Pi 3B+ (RPI3B+) is a typical IoT device. We follow the literature [53] to expand the TEE RAM (i.e., secure memory).

**Models and datasets.** We evaluate TensorShield with 4 typical DNN models that are widely used for device learning (i.e., MobileNetV2 [46], ResNet-18 [21], ResNet-50 [21], VGG16\_BN [50]). We use 4 popular datasets, i.e., CIFAR-10 [31], CIFAR-100 [31], STL-10 [11], and Tiny-ImageNet [32] datasets with resized input size 3 $\times$ 32 $\times$ 32, 3 $\times$ 32 $\times$ 32, 3 $\times$ 128 $\times$ 128, and 3 $\times$ 224 $\times$ 224. The MobileNetV2 is the simplest model and the VGG16\_BN is the most complex model in our evaluation. The CIFAR-10 and STL-10 are simple datasets while CIFAR-100 and Tiny-ImageNet are complex datasets. The dataset and model selection refers to prior secure inference literatures [37, 38, 41, 53, 75, 76].

**Baselines.** We compare the performance of TensorShield with the following 7 baselines, consisting of 1 no-shield (i.e., Native), 4 partial-shield (i.e., DarkneTZ [38], Serdab [16], Magnitude [22], and ShadowNet [53]), and 2 all-shield (i.e., GroupCover [76] and Occlumency [34]) solutions:

- **Native** executes the whole victim model in REE.
- **DarkneTZ** [38] shields victim model’s deep layers.
- **Serdab** [16] shields victim model’s shallow layers.
- **Magnitude** [22] shields victim model’s large magnitude weights.
- **ShadowNet** [53] shields non-linear layers and obfuscates linear layers in a fixed strategy.
- **GroupCover** [76] shields non-linear layers and obfuscates linear layers in a random strategy.

- **Occlumency** [34] shields all victim weights in TEE.

**Metrics.** There are two types of metrics in our evaluations. (1) Security metrics. We use the MS accuracy and MIA accuracy as model security metrics. The MS accuracy measures how many test samples can be correctly classified by the attacker’s surrogate model. Achieving high accuracy is a primary goal of model stealing attacks. The MIA accuracy represents the membership classification accuracy. (2) System metrics. For inference latency, we use the model inference time as our main metric which measures the time between feeding an input and getting the output. We also measure the energy consumption during the inference.

**Configuration.** For all cases, we use the public models as initialization to get a better model performance. We follow the hyperparameter settings of Knockoff Nets [41]. We use a minibatch size of 64, select cross-entropy loss, use SGD with weight decay of 5e-4, a momentum of 0.5, and train the victim models for 100 epochs. The learning rate is originally set to 0.1 and decays by 0.1 every 60 epochs. For the training in the MIA part, we follow the settings of ML-DOCTOR. The learning rate is 1e-2. The hyper-parameters to train shadow models are the same as victim models. We set the JS-divergence threshold as 0.1, a common value for evaluating two distributions [12, 35]. To ensure that the results on the device are not impacted by the throttle, we lock the CPU frequency and set sufficient time gaps between each test. We run all experiments ten times and record the average inference time.

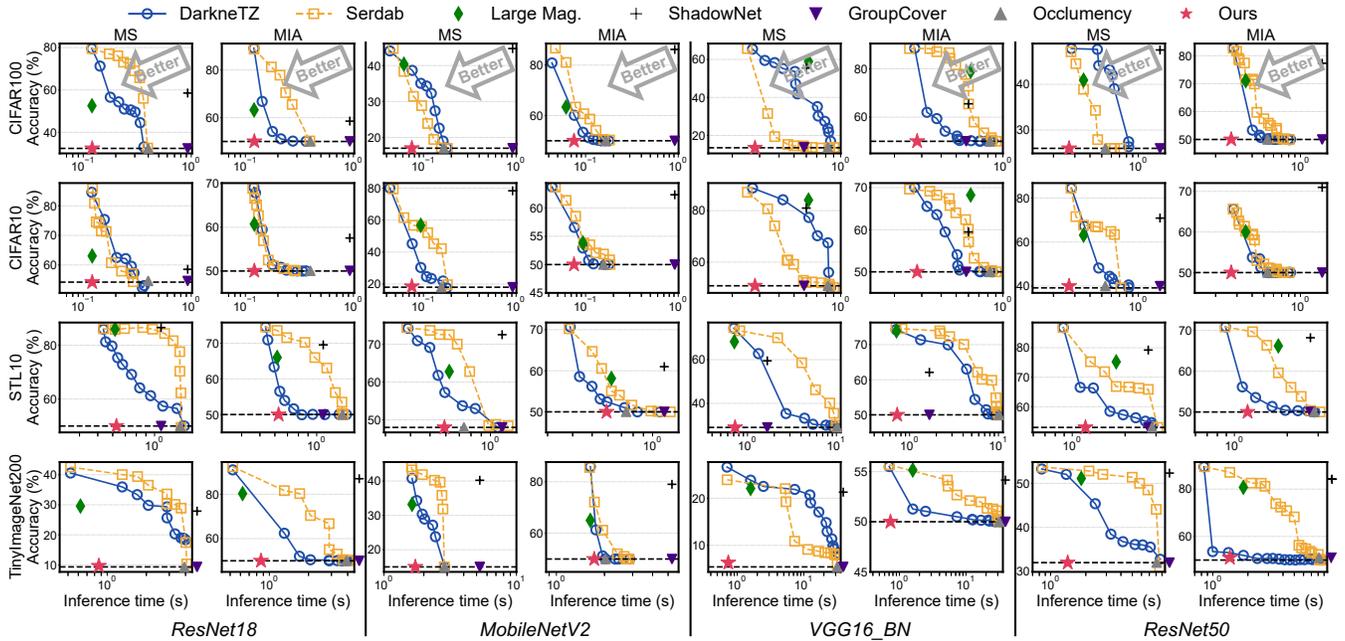
## 8.2 Overall performance

Figure 9 shows the overall results comparing TensorShield with all baselines. We follow the attack pipelines in §2.1. For DarkneTZ [38] and Serdab [16], we use the offloaded layers to replace the corresponding layers of  $\mathcal{M}_{init}$ . For Magnitude [22], we use the offloaded weights to replace the corresponding weights in  $\mathcal{M}_{init}$ . For ShadowNet [53], the attacker uses the public model to decode the obfuscation algorithm and uses the decoded weights to initialize  $\mathcal{M}_{init}$ . The accuracy of  $\mathcal{M}_{vic}$  used in the evaluation is shown in Table 2.

As shown in Figure 9, TensorShield achieves an inference speedup of up to 25.35 $\times$  (avg. 5.85 $\times$ ) and up to 16.89 $\times$  (avg. 4.32 $\times$ ) compared to GroupCover and Occlumency, respectively. Meanwhile, TensorShield achieves the same security protection as shielding the entire model inside TEE, i.e., 1.03 $\times$  for MS and 1.00 $\times$  for MIA. We can make the following observations:

(i) Under the same model architecture, when compared with shielding partial layers (i.e., Serdab and DarkneTZ) that achieve nearly identical inference latency as TensorShield, TensorShield offers better protection against MS in more complex datasets. For example, the first and last rows show better MS protection results than the second and third rows in Figure X. This is because complex datasets require both shallow and deep tensors in DNNs to perform comprehensive feature extraction, a task not achievable by tensors fixed at certain depths. TensorShield is able to identify critical tensors at varying depths to defend against MS attacks.

(ii) Within the same dataset, as model complexity increases, TensorShield provides better protection against MIA compared to shielding partial layers (i.e., Serdab and DarkneTZ) that achieve nearly identical inference latency as TensorShield. For example, for the STL10 dataset, TensorShield’s effectiveness in protecting



**Figure 9: Model stealing (MS) and membership inference attack (MIA) accuracy and inference time regarding baselines on Hikey960. The dotted line “- - -” represents black-box accuracy. The points of GroupCover and ShadowNet outside the right border of the subfigure represent OOM (i.e., out-of-memory error in TEE). TensorShield (Ours) shields critical tensors, achieving inference speeds up to 25.35× (avg. 5.85×) and 16.89× (avg. 4.32×) faster than GroupCover [76] and Occlumency [34], respectively.**

against MIA increases for MobileNetV2, ResNet18, ResNet50, and VGG16\_BN. This is because as the complexity of models increases, it becomes evident that not only the final classifier tensor harbors substantial membership privacy information, but the preceding tensors do as well. This indicates that for simple models, protecting only the final classifier layer is often sufficient. However, for more complex models, such protection is inadequate to satisfy security requirements against MIA.

(iii) Compared to shielding all layers (i.e., ShadowNet, GroupCover, and Occlumency), the more complex the model, the greater the improvement in inference performance by TensorShield. For instance, on the TinyImage200 dataset compared to Occlumency, TensorShield shows substantial performance improvements for VGG16\_BN, ResNet50, ResNet18, and MobileNetV2 by 16.89×, 5.78×, 4.54×, and 1.66×, respectively. This is because simpler models are more computationally suited to CPU processing, whereas more complex models are better handled by TensorShield, which could execute critical tensors in the REE by obfuscating them on the GPU.

(iv) The shapes of the curves for shielding deep layers (i.e., DarkneTZ) and shallow layers (i.e., Serdab) differ significantly across datasets and models. For a given victim model, setting a uniform threshold is challenging without comprehensive empirical measurements of both security and inference efficiency. This variability arises because the “sweet spots” for optimal shielding differ across datasets. For example, when shielding deep layers of ResNet18 to prevent model stealing (the first column in Figure X), the curve shapes for CIFAR-10 and CIFAR-100 differ markedly from those for STL-10.

(v) As the complexity of a dataset increases, the attack accuracy of MIA tends to rise while the model’s MS accuracy tends to decrease. This phenomenon can be attributed to the increased difficulty in comprehending the model’s decision-making processes when faced with complex datasets, which also heightens the propensity for overfitting. Such overfitting characterized by a lack of generalization, results in a more pronounced leakage of membership information.

**Understanding TensorShield’s improvements.** TensorShield outperforms various baselines for slightly different reasons. (i) Compared to the shielding none or partial layers (or weights) baselines, TensorShield has lower MS and MIA accuracy that achieves all-shield protection efficacy, because it dynamically selects critical tensors based on our critical tensor identification method (cf., §4) for different datasets and model architectures instead of fixed selection solutions. We will show the efficacy of our critical tensor evaluator in §8.3. Since fixed obfuscation is easy to reverse and restore victim weights, compared to ShadowNet, TensorShield still achieves higher model security. (ii) Compared to the all-shield baselines, TensorShield only masks privacy-related features (cf., §5) and plans tensor optimal execution strategy based on computation modeling (cf., §6), thus being able to reduce the inference latency. We will show details of runtime performance in §8.4.

### 8.3 Efficacy of Critical Tensor Identification

Next, we show the efficacy of critical tensor identification (cf. §4). We first compare TensorShield with two representative XAI methods [25, 55]: the Gradient method [55] utilizes integrated gradients,

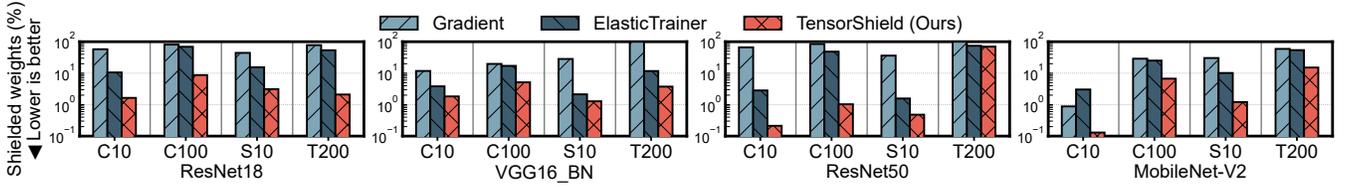


Figure 10: Comparison of representative XAI-based tensor selection methods.

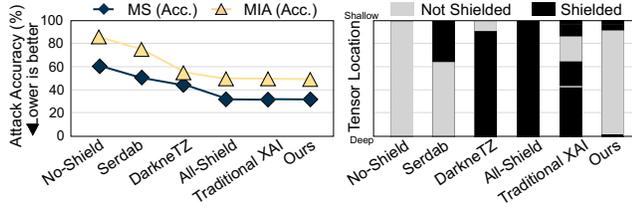


Figure 11: Attack accuracy (lines) regarding different defense schemes (bars) for ResNet18 with CIFAR100 dataset. We can only shield  $\sim 8\%$  weights (we show the selected tensors in Figure 12) within TEE to achieve comparable protection efficacy as shielding all weights.

whereas ElasticTrainer [25] represents state-of-the-art work combining both gradients and weight updates. To ensure a fair comparison, we evaluate the importance of tensors using these methods and then rank their importance values. Tensors are selected for protection from highest to lowest until they meet the specified MS accuracy threshold (i.e.,  $Acc_{MS}^{All} + 1\%$ ). As shown in Figure 10, TensorShield selects average 87.82% (absolutely 50.60%) fewer parameters compared to the Gradient method, and relatively 70.32% (absolutely 17.62%) fewer than ElasticTrainer. Notably, TensorShield exhibits superior performance on datasets with fewer classification tasks (e.g., CIFAR-10 and STL-10). This improvement can be attributed to the public pre-trained models carrying more decision-making information from simpler datasets. TensorShield effectively reduces the importance of these tensors, thus excluding them from selection (cf. §4.1).

To better understand the efficacy, we present in Figure 11 the MS accuracy of a ResNet18 model trained on the CIFAR100 dataset using different defense methods. We configure Serdab and DarkNeTZ to protect an identical number of layers (i.e., four layers). In contrast, traditional XAI methodologies employ ElasticTrainer to safeguard critical tensors. Our key observations include: (i) Although DarkNeTZ protects over 85% of the model parameters, it fails to secure the model effectively. In comparison, TensorShield adeptly identifies critical tensors distributed across both shallow and deep tensors in the victim model. (ii) Although finding important tensors via traditional XAI can achieve model security, the volume of parameters it selects far exceeds that of TensorShield. This is because important tensors are not necessarily critical tensors. The tensors selected by each method are illustrated in Figure 12. Notably, TensorShield (Figure 12(d)) excludes five important tensors compared to ElasticTrainer (Figure 12(c)). Specifically, TensorShield excludes three tensors because they perform as well as those in the pre-trained public model (cf. the attention transition term in

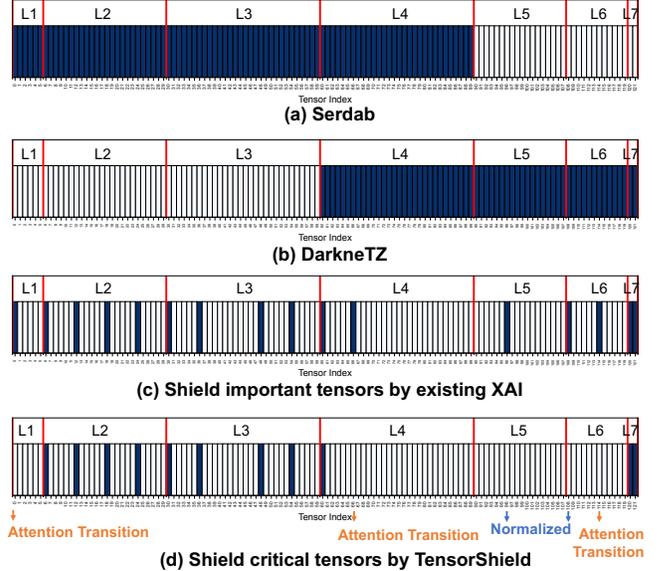


Figure 12: Tensor selections regarding different defense schemes for ResNet18 with CIFAR100 dataset. (a) Serdab: shields four shallow layers. (b) DarkneTZ: shields four deep layers. (c) Shielding important tensors by traditional XAI (i.e., ElasticTrainer). (d) TensorShield: shields critical tensors. The parameters are shown in Figure 11.

Eq. (1)). It also excludes two tensors due to their excessive parameter count, as determined by the intrinsic importance normalization term in Eq. (1). Additionally, the order of tensor criticality in TensorShield significantly diverges from the importance ranking used by ElasticTrainer, thereby demonstrating the superior efficacy of TensorShield’s critical tensor identification mechanism.

We visualize TensorShield’s critical tensor evaluation process in Figure 13. The decision-making attention of the conv1 tensor in the victim model is similar to that in the public model, whereas layer3.0.conv1 exhibits significant differences. This indicates that layer3.0.conv1 contains more critical information specific to the victim model. Therefore, as shown in Figure 13(right), TensorShield reduces the importance of the conv1 tensor while preserving the significance of layer3.0.conv1.

## 8.4 Runtime Performance

**Overall latency.** We conduct on-device inference experiments on the four models using three datasets (i.e., CIFAR-100, STL10, and Tiny-ImageNet). To ensure fairness, we compare TensorShield with

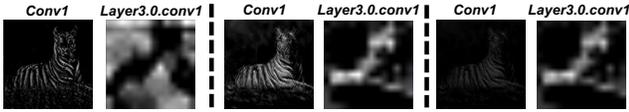


Figure 13: Visualization of critical tensor evaluation. Higher brightness indicates greater attention. *Left*: Public model’s attention. *Middle*: Victim model’s attention. *Right*: Critical tensor evaluation results.

baselines that achieve the black-box protection security effects (i.e., GroupCover and Occlumency). The results on two hardware platforms are shown in Figure 14. On mobile devices (i.e., Hikey960), TensorShield’s inference time is up to  $25.35\times$  (avg.  $5.85\times$ ) and  $16.89\times$  (avg.  $4.32\times$ ) faster than GroupCover and Occlumency, respectively. This latency speedup is due to TensorShield’s computation power modeling and the reduced overhead from masking, which we will detail in the following breakdown experiment. Even on IoT devices lacking GPU acceleration (i.e., Raspberry Pi 3B+), TensorShield achieves inference times that are up to  $2.53\times$  (avg.  $1.74\times$ ) and  $5.74\times$  (avg.  $2.11\times$ ) faster than GroupCover and Occlumency, respectively.

**Breakdown.** To better understand the reasons behind TensorShield’s speedup, we conduct a breakdown of inference times on the Hikey960, as shown in Table 3. The inference process for GroupCover includes calculation, communication, masking, and deobfuscation, while Occlumency’s process involves only calculation and decryption. We make two observations: (i) Depending on the size of the input, TensorShield adaptively chooses whether to execute tensors within the TEE or after obfuscating them to the REE. For example, for the CIFAR-100 dataset, due to the smaller computational demand, executing critical tensors in the TEE using the CPU is faster than after obfuscation on the GPU. Therefore, TensorShield computes critical tensors in the TEE without needing to mask intermediate features, whereas GroupCover obfuscates all parameters to the REE GPU and masks all intermediate features. Compared to Occlumency, TensorShield saves on the additional overhead of tensor computations and decryption in the TEE. For the STL-10 and Tiny-ImageNet datasets, which require heavier computations, GPU execution significantly outpaces CPU, hence TensorShield obfuscates tensors to the REE for GPU-accelerated execution, achieving a  $3.71\times$ - $9.82\times$  improvement in calculation time over Occlumency. (ii) TensorShield only masks the features of the last two tensors in the STL-10 dataset, reducing the masking time by 52.60% compared to GroupCover. This is attributed to the membership-aware masking technique.

## 8.5 Energy Consumption

We measure the energy consumption using a Deli DL333501C power meter [13] during the four model inference processes on two platforms. The results are shown in Figure 15. TensorShield achieves an average reduction in energy consumption of 58.66% (up to 91.35%) and 69.86% (up to 98.35%) compared to GroupCover and Occlumency, respectively, on the Hikey960. On the RPI3B+, it similarly reduces energy consumption by an average of 34.23% (up to 60.05%) compared to GroupCover and an average of 32.07% (up to 82.09%) compared to Occlumency. These improvements can be attributed to

Table 3: Inference time breakdown for ResNet18 model with CIFAR-100 dataset. "Cal", "Dec", "Comm", "Mask", and "Deobf" refer to calculation, decryption, communication, masking, and deobfuscation.  $\times$  represents OOM.

Dataset	Work	Execution time					Total
		Cal.	Dec.	Comm.	Mask.	Deobf.	
C100	Occlumency [34]	0.219s	0.080s	/	/	/	0.299s
	GroupCover [76]	0.497s	/	0.023s	0.403s	0.049s	0.926s
	<b>Ours</b>	<b>0.122s</b>	/	<b>0.002s</b>	/	/	<b>0.124s</b>
S10	Occlumency [34]	1.288s	0.080s	/	/	/	1.368s
	GroupCover [76]	0.510s	/	0.065s	0.481s	0.027s	1.083s
	<b>Ours</b>	<b>0.347s</b>	/	<b>0.037s</b>	<b>0.228s</b>	<b>0.016s</b>	<b>0.628s</b>
T200	Occlumency [34]	3.741s	0.080s	/	/	/	3.821s
	GroupCover [76]	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
	<b>Ours</b>	<b>0.381s</b>	/	<b>0.056s</b>	<b>0.364s</b>	<b>0.062s</b>	<b>0.863s</b>

Table 4: Total profiling times of two devices.

Device	Profiling time					
	CPU	GPU	Transfer	Masking	Obfuscating	Total
Hikey960 [2]	3min	3min	6min	6min	5min	23min
Raspberry Pi 3B+ [1]	18min	/	42min	1h 18min	5min	1h 25min

the reduced on-device inference time. Relative to native execution, TensorShield on the Hikey960 consumes, on average, 36.89% less energy than native CPU execution, but 7.07% more than native GPU execution. On the RPI3B+, it shows an average increase of 60.61% in energy consumption compared to native CPU execution.

## 8.6 Overhead

**Hardware profiling time.** We evaluate the overhead of TensorShield’s profiling time on two hardware platforms for computation modeling. The total profiling times for all models on each device are detailed in Table 4. RPI3B+’s profiling time is significantly longer than Hikey960 due to the limited computation power.

**Tensor Evaluating time.** Critical tensor evaluating is another major overhead in TensorShield. TensorShield requires a maximum of 20 epochs (average 10.9 epochs) compared to the standard evaluation method which uses 100 epochs. This results in an 89.1% reduction in evaluation time while maintaining equivalent assessment accuracy, showing the efficacy of our selection algorithm (cf. 4.2).

**Planning time.** Since the impact of consecutive tensors can be neglected, our optimization solution time is less than 10 seconds, which is negligible.

Notably, the profiling is a one-shot offline process for the specific platform, while the evaluating and planning stages are one-shot offline processes for the specific victim model.

## 9 DISCUSSION

**Supported models.** In this paper, we focus on CNN models because they are well-deployed on devices. For example, Xu et al. [67] comprehensively investigated 16,500 of the most popular Android apps on smartphones, and found that 87.7% of the models used in deep learning apps are CNN models. TensorShield can also be applied to other popular model architectures. For example, deploying Large Language Models (LLMs) on mobile devices has achieved great advances [7, 66, 68, 69], and the transformer-based model also shows that different component (sub-layer) in trained Transformer models have different importance [62]. Expanding TensorShield to optimize for LLMs will be our future work.

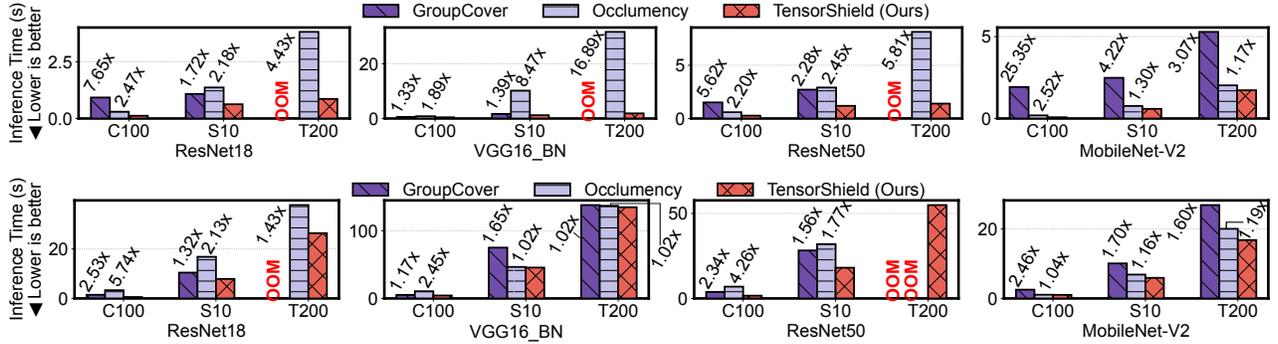


Figure 14: Inference time of 4 models compared with native and two completely secure baselines. *Top: Hikey960. Bottom: RaspberryPi 3B+.*

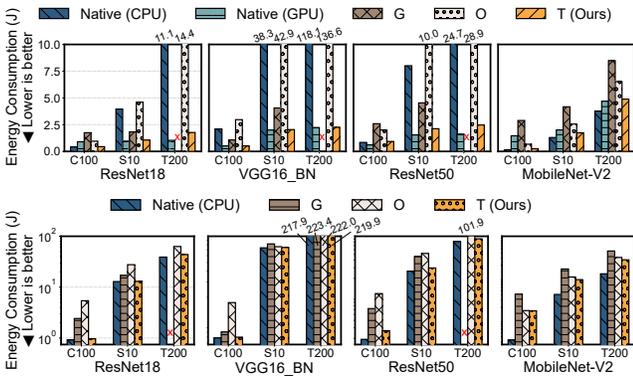


Figure 15: Comparison of native and all-shield solutions with energy consumption. *Left: Hikey960. Right: RPI3B+.* "G", "O", and "T" represent GroupCover, Occlumncy, and TensorShield (Ours), respectively. X represents OOM.

**New TEE architectures.** Despite ARM TrustZone has been widely used in the mobile and IoT industry, new TEE architectures (e.g., ARM CCA [5] and RISC-V KeyStone [33]) are still emerging. Although such new TEEs may mitigate the performance overhead of shielding the entire model solutions, they do not harm the practicality of TensorShield because the computation speed of such new TEEs is still not comparable with the device’s hardware accelerators (e.g., GPU). We believe TensorShield can be a promising solution to bridge the gap between the new TEEs and the evolving accelerators.

## 10 RELATED WORK

**On-device secure inference.** We show the related work in Table 5. To fully protect model security, some previous work [22, 34, 64] leverage TEE to shield the entire model to achieve black-box protection. They propose advanced techniques to reduce secure memory usage but execute all layers in the TEE where lose the opportunity to leverage GPU for acceleration. To speed up secure inference, existing research partitions the model and fixed part of the model to the TEE [16, 22, 38, 48]. Although they reduce inference latency, they cannot fully defend MS and MIA [75]. In order to enable the entire model to be accelerated by GPU, ShadowNet [53] and GroupCover [76] partition the DNN by the layer types and

Table 5: Comparison of TensorShield with related work

Works	Model Security		Shielding	Efficiency
	MS	MIA		
DarkneTZ [38]	●	●	deep layers	High
Serdab [16]	●	●	shallow layers	High
Magnitude [22]	●	●	large mag. weights	High
SOTER [48]	●	●	Inter. layers	High
ShadowNet [53]	●	●	non-linear layers	Medium
GroupCover [76]	●	●	non-linear layers	Medium
Occlumncy <sup>‡</sup> [34]	●	●	all layers	Low
<b>TensorShield (Ours)</b>	●	●	critical tensors	High

† ● stands for fully protection. ‡ It needs to encrypt weights offline.

shield non-linear layers using TEE. The offloaded linear layers are protected by lightweight obfuscation algorithms (e.g. matrix transformation) and all intermediate features are masked by OTP. The former utilizes a fixed obfuscation that has security issues while the latter utilizes random obfuscation achieving the black-box protection. However, both of them have high overheads of obfuscation and masking. Overall, none of the existing works simultaneously satisfy our requirements.

**Mitigating model attacks in training.** Existing defenses have explored different training strategies for migrating the model attacks. For example, Beowulf [18] mitigates MS attacks by reshaping the victim model’s decision regions to more complex and narrow. AMAO [29] proposed to use adversarial training to shield victim models from MS. SELENA [56] mitigates MIA by self-distillation by a novel ensemble architecture in the training process. TEESlice [75] is a recent work that inserts small external model slices into the public model and only trains the slices to force private weights located in the slices. Shielding these slices in the TEE can defend against MS and MIA. However, the scenario of TensorShield (and previous TEE-based secure inference solutions) is different from the above work, i.e., performing security analysis after the victim model is finished trained in an arbitrary strategy.

**TEE in GPUs.** Researchers have explored GPU TEEs to guarantee data security on GPUs. Implementing trusted architectures directly inside GPUs could achieve isolation [23, 58], but require customizing hardware (e.g., NVIDIA H100 GPU [40]) and are specific designed for cloud servers. Recent works have also proposed

several ARM-based GPU TEEs [15, 42, 59, 60]. However, our solution employs GPUs in an “out-of-the-box” manner, which requires no change to the hardware or shipped firmware for mobile and IoT devices.

## 11 CONCLUSION

In this paper, we present TensorShield, a new technique that selects critical tensors to fully defend against MS and MIA. TensorShield achieves up to  $25.35\times$  (avg.  $5.85\times$ ) speedup in inference latency without accuracy loss when compared to the state-of-the-art schemes, and also reduces energy consumption by an average of 58.66%.

## REFERENCES

- [1] 2016. Raspberry Pi 3 Model B. <https://www.raspberrypi.com/products/raspberrypi-3-model-b/>.
- [2] 2017. Hikey960. <https://www.96boards.org/product/hikey960/>.
- [3] 2018. Raspberry Pi 3 Model B+. <https://www.raspberrypi.com/products/raspberrypi-3-model-b-plus/>.
- [4] ARM. 2022. TrustZone for Cortex-A. <https://www.arm.com/technologies/trustzone-for-cortex-a>.
- [5] Arm. 2023. Arm Confidential Compute Architecture. <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>.
- [6] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. 2018. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In *Information Security Theory and Practice: 11th IFIP WG 11.2 International Conference, WISTP 2017, Heraklion, Crete, Greece, September 28–29, 2017, Proceedings 11*. Springer, 93–109.
- [7] Dongqi Cai, Shangguang Wang, Yaozong Wu, Felix Xiaozhu Lin, and Mengwei Xu. 2023. Federated few-shot learning for mobile nlp. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom'23)*, 1–17.
- [8] Dingfan Chen, Ning Yu, and Mario Fritz. 2022. Relaxloss: Defending membership inference attacks without losing utility. In *International Conference on Learning Representations (ICLR'22)*.
- [9] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, 395–412.
- [10] Hao-Jen Chien, Hossein Khalili, Amin Hass, and Nader Sehatbakhsh. 2023. Enc2: Privacy-Preserving Inference for Tiny IoTs via Encoding and Encryption. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom'23)*, 1–16.
- [11] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 215–223.
- [12] Jacob Deasy, Nikola Simidjievski, and Pietro Liò. 2020. Constraining variational inference with geometric jensen-shannon divergence. *Advances in Neural Information Processing Systems (NeurIPS'20)* 33 (2020), 10647–10658.
- [13] Deli. 2024. DL333501 Power Monitor. <https://www.delitoolsglobal.com/Power-Monitor-DL333501.html>.
- [14] Sen Deng, Mengyuan Li, Yining Tang, Shuai Wang, Shoumeng Yan, and Yinqian Zhang. 2023. {CipherH}: Automated Detection of Ciphertext Side-channel Vulnerabilities in Cryptographic Implementations. In *32nd USENIX Security Symposium (USENIX Security'23)*, 6843–6860.
- [15] Yunjie Deng, Chenxu Wang, Shunchang Yu, Shiqing Liu, Zhenyu Ning, Kevin Leach, Jin Li, Shoumeng Yan, Zhengyu He, Jiannong Cao, et al. 2022. Strongbox: A gpu tee on arm endpoints. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*, 769–783.
- [16] Tarek Elgarnal and Klara Nahrstedt. 2020. Serdab: An IoT Framework for Partitioning Neural Networks Computation across Multiple Enclaves. In *Proc. of IEEE/ACM CCGRID*, 519–528.
- [17] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom'18)*, 115–127.
- [18] Xueluan Gong, Rubin Wei, Ziyao Wang, Yuchen Sun, Jiawen Peng, Yanjiao Chen, and Qian Wang. 2024. Beowulf: Mitigating Model Extraction Attacks Via Reshaping Decision Regions. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS'24)*, 3838–3852.
- [19] Liwei Guo and Felix Xiaozhu Lin. 2022. Minimum viable device drivers for ARM TrustZone. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys'22)*, 300–316.
- [20] Peizhen Guo, Bo Hu, and Wenjun Hu. 2021. Mistify: Automating DNN Model Porting for On-Device Inference at the Edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*, 705–719.
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'16)*, 770–778.
- [22] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model Protection: Real-Time Privacy-Preserving Inference Service for Model Privacy at the Edge. *IEEE Transactions on Dependable and Secure Computing (TDSC)* 19, 6 (2021), 4270–4284.
- [23] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G Edward Suh. 2022. Guardnnc: secure accelerator architecture for privacy-preserving deep learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC'22)*, 349–354.
- [24] Kai Huang and Wei Gao. 2022. Real-time neural network inference on extremely weak devices: agile offloading with explainable AI. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom'22)*, 200–213.
- [25] Kai Huang, Boyuan Yang, and Wei Gao. 2023. Elastictrainer: Speeding up on-device training with runtime elastic tensor selection. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services (MobiSys'23)*, 56–69.
- [26] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security'20)*, 1345–1362.
- [27] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. 2022. CoDL: efficient CPU-GPU co-execution for deep learning inference on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys'22)*, 209–221.
- [28] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. 2019. Memguard: Defending against black-box membership inference attacks via adversarial examples. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security (CCS'19)*, 259–274.
- [29] Wenbo Jiang, Hongwei Li, Guowen Xu, Tianwei Zhang, and Rongxing Lu. 2023. A comprehensive defense framework against model extraction attacks. *IEEE Transactions on Dependable and Secure Computing* 21, 2 (2023), 685–700.
- [30] Dongwoo Kim and Cyril Guyot. 2023. Optimized privacy-preserving cnn inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2175–2187.
- [31] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
- [32] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.
- [33] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanovic. 2019. Keystone: A framework for architecting tees. *arXiv preprint arXiv:1907.10119* (2019).
- [34] Taegyong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom'19)*, 1–17.
- [35] Duo Li, Junxiang Bai, and Wenling Li. 2022. Multi-sensor Data Consistency and Fusion Based on Jensen-Shannon Divergence. In *International Conference on Guidance, Navigation and Control*. Springer, 5595–5605.
- [36] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. 2021. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security'21)*. USENIX Association, 717–732.
- [37] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. 2022. ML-Doctor: Holistic Risk Assessment of Inference Attacks against Machine Learning Models. In *31st USENIX Security Symposium (USENIX Security'22)*, 4525–4542.
- [38] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys'20)*, 161–174.
- [39] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. 2024. Machine learning with confidential computing: A systematization of knowledge. *ACM computing surveys (CSUR)* 56, 11 (2024), 1–40.
- [40] NVIDIA. 2022. NVIDIA CONFIDENTIAL COMPUTING. <https://www.nvidia.com/en-us/data-center/solutions/confidential-computing>.
- [41] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR'19)*, 4954–4963.
- [42] Heejin Park and Felix Xiaozhu Lin. 2023. Safe and Practical GPU Computation in TrustZone. In *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys'23)*, 505–520.

- [43] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–36.
- [44] Keegan Ryan. 2019. Hardware-backed heist: Extracting ECDSA keys from Qualcomm’s trustzone. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS’19)*. 181–194.
- [45] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. 2020. ML-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Proc. of 26th Annual Network and Distributed System Security Symposium (NDSS’19)*. 1345–1362.
- [46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR’18)*. 4510–4520.
- [47] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision (CVPR’17)*. 618–626.
- [48] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. 2022. SOTER: Guarding Black-box Inference for General Neural Networks at the Edge. In *2022 USENIX Annual Technical Conference (ATC’22)*. 723–738.
- [49] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. 2022. Model stealing attacks against inductive graph neural networks. In *2022 IEEE Symposium on Security and Privacy (S&P’22)*. IEEE, 1175–1192.
- [50] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. In *Proc. of 3rd International Conference on Learning Representations (ICLR’15)*.
- [51] Gowthami Somepalli, Liam Fowl, Arpit Bansal, Ping Yeh-Chiang, Yehuda Dar, Richard Baraniuk, Micah Goldblum, and Tom Goldstein. 2022. Can neural nets learn the same model twice? investigating reproducibility and double descent from the decision boundary perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR’22)*. 13699–13708.
- [52] Yu Sun, Gaojian Xiong, Jianhua Liu, Zheng Liu, and Jian Cui. 2024. TSQP: Safeguarding Real-Time Inference for Quantization Neural Networks on Edge Devices. In *2025 IEEE Symposium on Security and Privacy (S&P’25)*. IEEE Computer Society, 1–1.
- [53] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. 2023. ShadowNet: A Secure and Efficient On-Device Model Inference System for Convolutional Neural Networks. In *2023 IEEE Symposium on Security and Privacy (S&P’23)*.
- [54] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. 2021. 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)*. 1955–1972.
- [55] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning (ICML’17)*. PMLR, 3319–3328.
- [56] Xinyu Tang, Saeed Mahloujifar, Liwei Song, Virat Shejwalkar, Milad Nasr, Amir Houmansadr, and Prateek Mittal. 2022. Mitigating membership inference attacks by {Self-Distillation} through a novel ensemble architecture. In *31st USENIX Security Symposium (USENIX Security 22)*. 1433–1450.
- [57] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *Proc. of International Conference on Learning Representations (ICLR’19)*.
- [58] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *Proc. of 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI’18)*. 681–696.
- [59] Chenxu Wang, Fengwei Zhang, Yunjie Deng, Kevin Leach, Jiannong Cao, Zhenyu Ning, Shoumeng Yan, and Zhengyu He. 2024. CAGE: Complementing Arm CCA with GPU Extensions. In *Network and Distributed System Security Symposium (NDSS’24)*.
- [60] Jinwen Wang, Yujie Wang, and Ning Zhang. 2023. Secure and timely gpu execution in cyber-physical systems. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS’23)*. 2591–2605.
- [61] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. 2021. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom’21)*. 215–228.
- [62] Wenxuan Wang and Zhaopeng Tu. 2020. Rethinking the Value of Transformer Components. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING’20)*. 6019–6029.
- [63] Huizi Xiao, Qingyang Zhang, Qingqi Pei, and Weisong Shi. 2021. Privacy-preserving neural network inference framework via homomorphic encryption and sgx. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS’21)*. IEEE, 751–761.
- [64] Xueshuo Xie, Haoxu Wang, Zhaolong Jian, Tao Li, Wei Wang, Zhiwei Xu, and Guiling Wang. 2024. Memory-Efficient and Secure DNN Inference on TrustZone-enabled Consumer IoT Devices. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM’24)*.
- [65] Guoliang Xing, Tian Wang, Zhihui Xie, and Weijia Jia. 2008. Rendezvous planning in wireless sensor networks with mobile elements. *IEEE Transactions on Mobile Computing* 7, 12 (2008), 1430–1443.
- [66] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. 2024. FwdLLM: Efficient Federated Finetuning of Large Language Models with Perturbed Inferences. In *2024 USENIX Annual Technical Conference (USENIX ATC’24)*. 579–596.
- [67] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference (WWW’19)*. 2125–2136.
- [68] Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. 2024. PowerInfer-2: Fast Large Language Model Inference on a Smartphone. *arXiv preprint arXiv:2406.06282* (2024).
- [69] Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. 2024. Llm as a system service on mobile devices. *arXiv preprint arXiv:2403.11805* (2024).
- [70] Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. 2022. Infi: end-to-end learnable input filter for resource-efficient mobile-centric inference. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom’22)*. 228–241.
- [71] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. 2024. CipherSteal: Stealing Input Data from TEE-Shielded Neural Networks with Ciphertext Side Channels. In *2025 IEEE Symposium on Security and Privacy (S&P’25)*. 79–79.
- [72] Yuanyuan Yuan, Zhibo Liu, Sen Deng, Yanzuo Chen, Shuai Wang, Yinqian Zhang, and Zhendong Su. 2024. 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’24)*. 4346–4360.
- [73] Chiyuan Zhang, Samy Bengio, and Yoram Singer. 2022. Are all layers created equal? *Journal of Machine Learning Research* 23, 67 (2022), 1–28.
- [74] Qiyang Zhang, Xiangying Che, Yijie Chen, Xiao Ma, Mengwei Xu, Schahram Dustdar, Xuanzhe Liu, and Shangguang Wang. 2023. A comprehensive deep learning library benchmark and optimal library selection. *IEEE Transactions on Mobile Computing* (2023).
- [75] Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. 2024. 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 (S&P’24)*. 52–52.
- [76] Zheng Zhang, Na Wang, Ziqi Zhang, Yao Zhang, Tianyi Zhang, Jianwei Liu, and Ye Wu. 2024. 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’24)*.