No Privacy Left Outside: On the (In-)Security of TEE-Shielded DNN Partition for On-Device ML

Ziqi Zhang*, Chen Gong†, Yifeng Cai*, Yuanyuan Yuan‡,
Bingyan Liu§, Ding Li*, Yao Guo*, Xiangqun Chen*
*Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University
†School of Computer Science, Peking University
‡The Hong Kong University of Science and Technology,
§School of Computer Science, Beijing University of Posts and Telecommunications
{ziqi_zhang,gongchen17,caiyifeng,ding_li,yaoguo,cherry}@pku.edu.cn,
yyuanaq@cse.ust.hk, bingyanliu@bupt.edu.cn

***Abstract*—On-device ML introduces new security challenges: DNN models become white-box accessible to device users. Based on white-box information, adversaries can conduct effective model stealing (MS) against model weights and membership inference attack (MIA) against training data privacy. Using Trusted Execution Environments (TEEs) to shield on-device DNN models aims to downgrade (easy) white-box attacks to (harder) black-box attacks. However, one major shortcoming of TEEs is the sharply increased latency (up to 50×). To accelerate TEE-shield DNN computation with GPUs, researchers proposed several model partition techniques. These solutions, referred to as TEE-Shielded DNN Partition (TSDP), partition a DNN model into two parts, offloading[1] the privacy-insensitive part to the GPU while shielding the privacy-sensitive part within the TEE. However, the community lacks an in-depth understanding of the seemingly encouraging privacy guarantees offered by existing TSDP solutions during DNN inference. This paper benchmarks existing TSDP solutions using both MS and MIA across a variety of DNN models, datasets, and metrics. We show important findings that existing TSDP solutions are vulnerable to privacy-stealing attacks and are *not* as safe as commonly believed. We also unveil the inherent difficulty in deciding the optimal DNN partition configurations, which vary across datasets and models. Based on lessons harvested from the experiments, we present TEESLICE, a novel TSDP method that defends against MS and MIA during DNN inference. Unlike existing approaches, TEESLICE follows a partition-before-training strategy, which allows for accurate separation between privacy-related weights from public weights. TEESLICE delivers the same security protection as shielding the entire DNN model inside TEE (the "upper-bound" security guarantees) with over $10\times$ less overhead (in both experimental and real-world environments) than prior TSDP solutions and no accuracy loss. We make the code and artifacts publicly available on the Internet.**

1. In this paper, "offload" refers to executing computation-intensive DNN operations on insecure devices with strong computation ability (*e.g.* GPUs), rather than secure devices with weak computation ability (*e.g.* TEEs).

1. Introduction

On-device machine learning (ML) has become an important paradigm for latency- and privacy-sensitive tasks [93], [110], [28], [92] on mobile and IoT devices. However, on-device learning also introduces new security threats to the deployed deep neural network (DNN) models: by making *DNN models white-box accessible to device users*, adversaries can obtain full model information and easily achieve high attack accuracy with much less cost for representative attacks like **Model Stealing** (MS) and **Membership Inference Attack** (MIA) [42], [77], [46], [80], [19], [57]. Therefore, one key objective for hardening on-device ML is to **prevent** adversaries from accessing the on-device models, thereby **downgrading** white-box MS and MIA attacks to black-box (much harder) settings [42], [68], [40], [92].

Unfortunately, protecting on-device DNN models is particularly challenging due to the security-and-utility trade-off. Algorithmic-level protections, such as Multi-Party Computation (MPC) [48], Homomorphic Encryption (HE) [32], Regularization [72], and Differential Privacy (DP) [29], are not applicable since they are either too computationally expensive for mobile and IoT devices, or downgrade the accuracy of the protected models significantly [42], [94]. Employing Trusted Execution Environments (TEEs) [37], [56], [50], [59] to directly host DNN models is also not practical, because shielding the whole DNN model in TEEs (shielding-whole-model) leads to about $50\times$ deduction in model speed due to TEE's limited computation speed [94].

While protecting an entire DNN model with TEEs is infeasible for on-device ML, recent works have advocated to protect privacy-related portion of a DNN model to offer high utility and security. In particular, researchers propose *TEE-shielded DNN partition* (TSDP), which only puts a subset of the DNN model in TEEs and offloads the rest computation on GPUs [68], [40], [85], [92]. Existing research broadly assumes that the offloaded part is insufficient to expose critical private information of DNN models, meaning the exposed model parts do not leak substantially more information than a black-box interface. However, we argue

that this assumption is questionable given a practical threat model where adversaries have access to abundant public information on the Internet, such as pre-trained models and public datasets [24], [22], [99]. To exploit TSDP, adversaries may use public information to analyze the offloaded model parts and acquire more privacy than only analyzing black-box outputs, breaking the promises of TSDP. Nevertheless, none of existing approaches have systematically evaluated their security promise when taking public information into account.

This paper conducts the first systematic empirical evaluation on the security of TSDP approaches. We first investigate papers published between 2018–2023 in prestigious venues, including IEEE S&P, MobiSys, ATC, ASPLOS, RTSS, MICRO, AAAI, ICLR, MICRO, and TDSC. We then put the reviewed approaches into five categories based on their technical pipelines, and empirically evaluated each category with MS/MIA initiated by a practical adversary with public information on hand.

The experiment shows that existing TSDP approaches expose substantial private information to attackers via the offloaded model weights, enabling approximately white-box quality attacks toward TEE shielded models. MS attack accuracies toward existing TSDP solutions are $3.76\times$ – $3.92\times$ higher than the black-box (shielding-whole-model) baseline. For comparison, the unprotected white-box baseline (offloading an entire DNN model outside TEE) has a $4.24\times$ higher accuracy than the shielding-whole-model setting. The results for MIA are similar. Existing TSDP approaches have $1.16\times$ – $1.28\times$ higher MIA accuracy in comparison to the shielding-whole-model baseline, while the accuracy for the white-box setting is $1.36\times$ higher.

Worse still, we encountered high difficulties to augment the security of existing TSDP approaches without making substantial changes to their methodologies. To illustrate this, we measured the attack accuracy of MS/MIA using various configurations of existing TSDP approaches. We found it particularly difficult to determine a "sweet spot" configuration, that can maximize a DNN model's utility while still satisfying security requirements. Specifically, given a maximally tolerant attack accuracy, existing TSDP approaches require substantially different settings to config the shielded part when protecting different models and datasets. Therefore, for all existing TSDP approaches, we need to conduct an empirical procedure to identify the "sweet spot" configuration for specific models and datasets. However, such empirical procedures are prohibitively expensive, given the vast number of potential model and dataset combinations.

The fundamental weakness of existing TSDP approaches is that they follow a *training-before-partition* strategy. This involves first training a private model with a public pre-trained model and private data, and then separating the model into two parts: a shielded part that runs in TEEs, and an offloaded part that runs out of TEEs. Since training occurs before model partition, privacy-related weights may likely pervade the entire model. Thus, it is hard for existing TSDP solutions to accurately *isolate* privacy-related weights, creating potential attack surfaces.

To ensure the security of TSDP solutions, we propose a slicing-based approach, called TEESLICE, that accurately *isolates* privacy-related weights from offloaded weights at the inference stage. TEESLICE adopts a *partition-before-training* strategy, which first partitions a DNN model into a backbone and multiple private slices, then reuses public pre-trained models as the backbone, and at last trains the slices with private data. Therefore, TEESLICE accurately separates privacy-related weights from offloaded weights and is able to shield *all* privacy-related weights in TEEs.

The key challenge to realizing the partition-before-training strategy is to ensure that the private slices are small enough to run in TEEs while maintaining a decent accuracy. To this end, we propose a dynamic pruning algorithm that first trains the private slices with large sizes that have a sufficient model capacity for high accuracy, and then optimizes the size of the slices automatically under a threshold of accuracy loss. In this way, TEESLICE automatically finds the "sweet spot" configuration, which minimizes the number of slices (computation) inside TEE while keeping the same accuracy as the corresponding unpartitioned model.

Our evaluation shows that TEESLICE outperforms existing TSDP approaches in terms of security guarantee and utility cost. Attackers can hardly obtain any private information by analyzing the model architectures, and as a result, TEESLICE is shown to achieve the security level of shielding-whole-model baseline with $10\times$ less computation cost (in both experimental and real-world environments) than other TSDP solutions. Besides, TEESLICE reaches a high-security level with nearly no sacrifice. The statistical evaluation shows no change between the accuracy of the TEESLICE protected model and the original unpartitioned model. The offloaded public backbone does not increase the attack performance as well. The contribution of this paper can be summarized as follows:

- We systematically evaluate the security guarantee of prior TSDP solutions using two representative attacks, MS and MIA, and reveal the security issues of these solutions.
- We illustrate the difficulty of improving the security of prior TSDP approaches without substantially changing their methodologies.
- We propose TEESLICE, a novel TSDP solution for DNN inference that isolates privacy from offloaded model parts to provide a strong security guarantee using TEEs and cryptographic primitives. Our thorough evaluation shows that TEESLICE offers a high security guarantee with moderate overhead and no accuracy loss.

Availability: The artifact is available at [2]. We also provide supplementary experimental results of this paper at [5].

2. Background and Threat Model

2.1. Background

TEE. A Trusted Execution Environment (TEE) is an isolated hardware enclave that stores and processes sensitive data.

Popular TEE implementations include Intel SGX [62], AMD SEV [49], and TrustZone [12]. In this paper, we follow prior work and deem TEE as *a secure area on a potential adversary host device (including GPUs)* [68], [40], [86], [92]. It means *the data, code, and the whole computation process* inside TEEs are secure. Although there are side-channel attacks that may leak sensitive data from TEE, they are out of our consideration.

TSDP Solutions. TSDP solutions aim to provide a black-box label-only protection against MS/MIA by shielding partial DNN models inside TEEs. The motivation is to reduce inference latency of the straightforward black-box protection that shields the whole model inside TEEs (increase latency by up to 50× [94]). The security goal of TSDP solutions is to *downgrade white-box MS/MIA against on-device models to black-box label-only attacks* [68], [40], [92], [85]. Such degeneration is important and practical for deployed DNN models in production environments. For MS, TSDP solutions enforce accurate, cheap (usually taking negligible number of queries) white-box attacks [110], [82] to expensive (usually taking tens of thousands of queries) and inaccurate black-box attacks [77]. For MIA, TSDP solutions provide a deployment framework to guarantee differential privacy requirements with little accuracy sacrifice [42].

2.2. Threat Model

Defender's Goal. The goal of this paper (and prior TSDP solutions) is to degrade white-box attacks to black-box label-only attacks. We consider the security guarantee of a black-box baseline, where TEE shields the whole DNN model and only returning prediction labels, as the *upper bound* security protection offered by TSDP approaches [68], [40], [85], [92]. We however do *not* aim to completely mitigate information leakage from TEE outputs (i.e., prediction labels).

Model Output. Following prior TSDP papers and also real-world productions [68], [40], [85], [92], we assume that the deployed models only generate labels to users (i.e., "label-only outputs"). In other words, we assume that the confidence of the model prediction is an intermediate result, therefore, can be protected by TEEs. This assumption is supported by a comprehensive survey on the output type of on-device ML systems [93]. Further, we also surveyed the eight most important on-device ML tasks. For each task, we collect the three most downloaded Android applications (24 apps in total) over three different application markets (Google Play, Tencent My App, and 360 Mobile Assistant). We manually checked the output type of the applications and found that *all* of the 24 applications only return the prediction label and keep the confidence of models in the intermediate results.

Defender/Adversary's Capability. We assume that both the model owner (defender) and the attacker can use the public model on the Internet [99], [24], [79] to improve the accuracy of the model or attacks, a realistic setting for modern on-device learning tasks [64], [33], [34], [65], [111], [27], [99], [24]. The attacker can infer the architecture of the whole protected model, or an equivalent one, based on the public information, such as the inference results or the unprotected model part, with existing techniques [24], [22], [40], [68], [38]. Besides, we assume that the attacker can query the victim model for limited times (less than 1% of the training data), a practical assumption shared by related work [82], [44], [103]. For simplicity, we denote the victim model as M_{vic}, the public model as M_{pub}, and the surrogate model produced by model stealing as M_{sur}.

3. Evaluating Existing TSDP Defenses

We first conduct a thorough literature review on recent publications in top conferences and journals and identify five categories of TSDP techniques. Then, we implement a representative technique from each category and evaluate their security via MS and MIA. We assess if they were sufficiently secure against the launched attacks, and we harvest empirical observations to present lessons from the evaluation.

TABLE 1: A taxonomy of existing TSDP solutions. We mark representative works empirically assessed in this study. Other works are ignored in our empirical evaluation and are just part of the literature review.

Literature	Conference/Journal	Category
DarkneTZ [68]	MobiSys 2020	Shielding Deep Layers
PPFL [67]	MobiSys 2021	
Shredder [66]	ASPLOS 2020	
Yerbabuena [36]	Arxiv 2018	Shielding Shallow Layers
Serdab [30]	CCGRID 2020	
Origami [71]	Arxiv 2019	
Magnitude [40]	TDSC 2022	Shielding Large-Mag. Weights
AegisDNN [102]	RTSS 2021	Shielding Intermediate Layers
SOTER [85]	ATC 2022	
ShadowNet [92]	S&P 2023	Shielding Non-Linear Layers & Obfuscation
DarKnight [38]	MICRO 2021	-
Goten [74]	AAAI 2021	-
Slalom [94]	ICLR 2018	-
GINN [14]	CODASPY 2022	-
eNNclave [84]	AISec 2020	-

3.1. Literature Summary

Defense Taxonomy. We identify publications that partition DNNs across TEEs and GPUs for privacy-preserving ML from leading conferences and journals from the past five years. Table 1 lists 15 important works from computer security, computer systems, mobile computing, artificial intelligence, and computer architecture.

Five papers do not meet the requirements under our threat model. Among them three are unable to defend models against MS (DarKnight [38], Slalom [94], and GINN [14]), one has a stronger assumption that requires two TEEs to verify each other (Goten [74]), and the last one decreases DNN accuracy significantly (eNNclave [84]; details in Sec. 6.2). We then divide the remaining ten papers into five categories depending on the TSDP schemes. Fig. 1

schematically illustrates each category using a sample 4-layer DNN model (including two convolution layers and two ReLU layers). We also include our proposed approach (details in Sec. 5) for comparison. The five categories are as follows:

① Shielding Deep Layers partitions the DNN according to the layer depth and places the layers close to the output layer in the TEE. In Fig. 1, two deepest layers (Conv2 and ReLU2) are shielded.

② Shielding Shallow Layers partitions the DNN according to the layer depth and places the layers close to the input layer in the TEE. In Fig. 1, two shallowest layers (Conv1 and ReLU1) are shielded.

③ Shielding Large-Magnitude Weights partitions the DNN according to the absolute weight value, and then puts the weights with large magnitudes and ReLU layers in the TEE. Fig. 1 shields partial convolution layers (to represent large-magnitude weights) and ReLU layers.

④ Shielding Intermediate Layers puts randomly chosen intermediate layers in the TEE. Fig. 1 shields ReLU1 and Conv2 as the random-selected layers.

⑤ Shielding Non-Linear Layers and Obfuscation partitions the DNN by the layer types and shields non-linear (*e.g.* ReLU) layers using TEE. The offloaded linear layers (*e.g.* convolution layers) are protected by lightweight obfuscation algorithms (*e.g.* matrix transformation). Fig. 1 shields the ReLU layers and offloads all convolution layers.

3.2. Representative Defenses

Scheme Selection. For each of the five TSDP schemes, we select one representative solution for evaluation. For shielding deep layers (①), we choose DarkneTZ because according to related work [38], [67], it is the state-of-the-art (SOTA) solution for protecting training data privacy. For shielding shallow layers (②) and shielding intermediate layers (④), we select Serdab and SOTER because they are the most recent papers published in peer-reviewed conferences. For shielding large-magnitude weights (③) and shielding non-linear layers (⑤), we choose Magnitude [40] and ShadowNet [92] since they are the only solutions in their respective categories.

Configuration Setting. All these schemes require configurations, e.g., for ①, we need to configure the exact number of "deep" layers in the TEE. Overall, we configure each defense scheme according to their papers. In particular, for DarkneTZ (①), we put the last classification layer into TEE. For Serdab (②), the TEE shields the first four layers. For Magnitude (③), the TEE shields 1% weights with the largest magnitudes. For SOTER (④), the TEE shields 20% randomly-selected layers and multiplies the other offloaded layers with a scalar to conceal the weight values. For ShadowNet (⑤), the TEE shields all the ReLU layers and obfuscates all the offloaded convolution layers with matrix transformation and filter permutation (detailed description in our Website [5]). Furthermore, we note that selecting proper configurations constitutes a key factor that undermines their security/utility guarantee. We will discuss other configurations later in Sec. 4.

3.3. Evaluated Attacks

Attack Selection. We consider MS and MIA attacks that can extract confidential model weights and private training data as the security benchmark for existing TSDP approaches. For MS, we employ standard query-based stealing techniques where the attacker trains a model from a set of collected data labeled by the partially-shielded M_{vic}. Query-based MS has been widely adopted in literature [77], [46], [86], [78]. We leverage the attack implementation offered by Knockoff Net [9], a SOTA baseline accepted by prestigious publications [78], [46], [86]. For MIA, we chose the transfer attack that is designed against the label-only scenario [60]. Transfer attack builds M_{sur} to imitate the behavior of M_{vic} and infer the privacy of M_{vic} from white-box information of M_{sur} (*e.g.* , confidence scores, loss, and gradients). The intuition is that membership information can transfer from M_{vic} to M_{sur}. We chose the standard confidence-score-based algorithm to extract membership from M_{sur}. In particular, this process trains a binary classifier, such that given the confidence score of M_{sur}, the classifier predicts if the corresponding DNN input is in the training data of M_{vic}. Confidence-score-based MIA has been extensively used in previous attacks [80], [83], [88], [60], [72], and we reused the attack implementation from a recent benchmark suite, ML-DOCTOR [10], [61]. Recent work has consistently employed ML-DOCTOR for membership inference [86], [23], [18].

Attack Pipeline. As in Fig. 2, the attack goal is to get the victim model M_{vic}'s functionality and membership information. The attack pipeline consists of three phases: surrogate model initialization (P_i), MS (P_{ii}), and MIA (P_{iii}). The attacker first analyzes the target defense scheme and then conducts P_i to get an initialized surrogate model (denoted as M_{init}). P_{ii} trains M_{init} with queried data and outputs the surrogate model M_{sur} (the recovered victim model). P_{iii} takes M_{sur} as input, uses MIA algorithms and outputs M_{vic}'s membership privacy. Specifically, in P_{iii}, the adversary first trains a binary classifier. Then, given an input i and the M_{sur}'s prediction confidence score p, the classifier decides if i belongs to the M_{vic}'s training data by taking p as its input [88], [18], [104].

Surrogate Model Initialization. Steps in our attack pipeline are automated except for the first step, surrogate model initialization (P_i). To run the attacks, attackers first need to construct M_{init} with the exposed knowledge in the public part of the TSDP protected models. The high-level process to construct M_{init} has three steps. First, the attacker infers the architecture of the TSDP protected model based on the offloaded part and the model output with existing techniques [24], [22]. Then, the attacker chooses a public model M_{pub} (with the same or an equivalent architecture) as M_{init}. Lastly, the attacker transports the model weights in the offloaded part of M_{vic} to the corresponding parts of M_{init}. For DarkneTZ (①), Serdab (②), and SOTER (④), we use

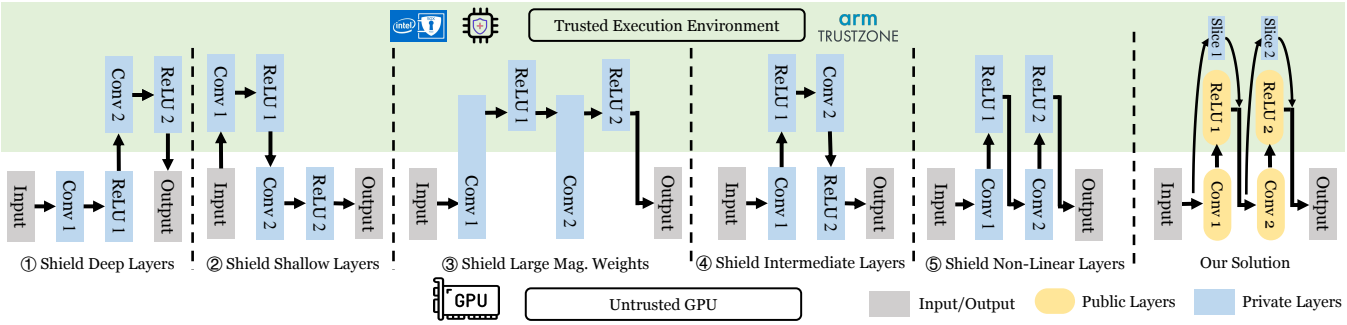

Figure 1: An illustration of different TSDP solutions on a four-layer DNN. Blue squares are privacy-related layers, and yellow rounded squares are privacy-irrelevant (public) layers. ① shields two deep layers (Conv2 and ReLU2) and ② shields two shallow layers (Conv1 and ReLU1). ③ shields the large-magnitude weight of each layer. ④ shields two random intermediate layers (ReLU1 and Conv2). ⑤ shields non-linear layers (ReLU1 and ReLU2) and obfuscates other layers (Conv1 and Conv2). Our solution (introduced in Sec. 5) shields privacy-related slices and non-linear layers of the public backbone model.

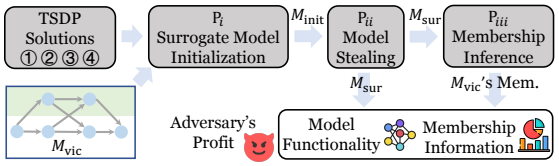

Figure 2: A three-phase attack pipeline.

the offloaded layers to replace the corresponding layers of M_{init}. For Magnitude (③), we use the offloaded weights that run on GPUs to replace the corresponding weights in M_{init}. For ShadowNet (⑤), the attacker uses the public model to decode the obfuscation algorithm (detailed description in our Website [5]) and uses the decoded weights to initialize M_{init}.

Comparison Baselines. To ease the comparison, we also provide baseline evaluation results. Notably, for MS, we consider the white-box solution (offloading the whole M_{vic} to GPU, referred to as "No-Shield") as the easiest baseline because the adversary can directly use the offloaded M_{vic} as M_{sur} and does not need to train the model. We consider a black-box (or shielding-whole-model) setting (referred to as "Black-box") where attackers can only access the model prediction labels and identify the M_{vic}'s corresponding public models. However, the attacker cannot leverage the M_{vic}'s weights offloaded on GPU to construct M_{init}. This is a challenging setting, and the attacker needs to steal every layer's weights from M_{vic}. As for MIA, the white-box setting denotes that the attackers can directly use the M_{vic}'s output confidence score for membership inference because the whole M_{vic} is offloaded. Recall, as noted in our threat model (Sec. 2), TSDP solutions often refuse to return confidence scores and only return predicted labels to mitigate membership inference. In contrast, the black-box setting denotes that the attacker directly uses M_{pub} as M_{init} and trains M_{sur} from queried data. Apparently, this is comparable to "random guess" (success rate around 50%[2]), given that a M_{init} (*i.e.* , M_{pub}) contains no information on the M_{vic}'s private training datasets.

2. In the evaluation setting (Sec. 3.4), we follow prior MIA work to set the size of target training dataset equal the size of target testing dataset.

3.4. Evaluation Setting

Datasets. We use four different datasets and five DNN models to evaluate different defenses. The dataset and model selection refers to prior MS and MIA literatures [61], [46]. In particular, the datasets include CIFAR10, CIFAR100 [52], STL10 [25], and UTKFace [109]. CIFAR10 and CIFAR100 are default choices of prior MS/MIA literatures [78], [46], [61], [18], [104]. STL10 and UTKFace are used by ML-DOCTOR to quantify model vulnerability [61]. When evaluating MS, we use each dataset's default train/test split to avoid possible bias. To evaluate MIA, we follow the setting of ML-DOCTOR to partition each dataset into four splits: the target training dataset, the target testing dataset, the shadow training dataset, and the shadow testing dataset. The model owner uses the target training and the target testing datasets to train and evaluate the victim model M_{vic}. The adversary uses the shadow training dataset and the shadow testing dataset to train the binary classifier. This is a common setting for evaluating MIA in prior work [18], [88], [86], [60].

Models. The benchmark models include ResNet18 [39], VGG16_BN [89], AlexNet [53], and ResNet34. These models are widely used in prior security studies [61], [24], [104], [82]. We mainly report the results on AlexNet, ResNet18, and VGG16_BN and leave the results of ResNet34 and VGG19_BN in the website [5]. We set the hyper-parameters following the paper and released code of prior works [10], [9]. As introduced in Fig. 2, for all cases, we use the public models as initialization to get a better model performance [77], [78]. For the training in the MS part, we follow the hyper-parameter settings of Knockoff Nets [9]. Accuracies of the trained models M_{vic} are reported in Table 5. The accuracies are generally consistent with public results [26], [54], [100]. For the training in the MIA part, we follow the settings of ML-DOCTOR [10]. Model accuracies are reported in the Website [5]. All models achieve high accuracy on the target training dataset, and the accuracies are

consistent with prior works [61]. We leave detailed settings of hyper-parameters in our Website [5].

Metrics. Overall, we systematically evaluate the effectiveness of de facto TSDP solutions using three MS metrics and four MIA [46], [82], [73], [90], [104]. Specifically, we record MS accuracy, fidelity, and attack success rate (ASR) according to prior work [46], [77], [110], [82]. For MIA, we use confidence-based MIA accuracy, gradient-based MIA accuracy, generalization gap, and confidence gap following prior literature as well [73], [90], [104]. Due to limited space, in the main paper, we only report "MS accuracy" (denoted as "Model Stealing") and "confidence-based MIA attack accuracy" (denoted as "Membership Inference") as the main metrics. We report the other metrics in the website [5]. The findings are consistent with the main paper. MS accuracy denotes the prediction accuracy of the M_{sur} [110], [77], [46], [86]. A higher accuracy indicates that M_{sur} successfully steals more functionality from M_{vic}. As for the MIA accuracy, a higher accuracy denotes that attackers can more accurately decide if a given sample is in M_{vic}'s training dataset.

3.5. Attack Results

We aim to answer the following research question primarily:

> **RQ1:** How secure are the selected TSDP solutions against model and data privacy stealing?

We report the evaluation results over three models (AlexNet, ResNet18, and VGG16_BN) in Table 2 (total 12 cases). As aforementioned, we also report the baseline settings ("No-Shield" and "Black-box") for comparison. To compare the performance between different defenses, for each case, we compute a relative accuracy as the times of the accuracy over the accuracy of the black-box baseline. We report the average relative accuracy in the last row of Table 2. A defense scheme is considered more effective if its corresponding attack accuracy is closer to the black-box baseline (the relative accuracy is closer to 1.00×). As Table 2 shows, for the white-box baseline (the whole M_{vic} is offloaded to the GPU), the relative accuracies are 4.26× for model stealing and 1.39× for membership inference.

From Table 2, we can observe that the defense effectiveness of all solutions is limited. It is evident that even the lowest attack accuracy (marked in yellow), indicating the highest defense effectiveness, among each setting, is still *much higher* than the black-box baseline: the lowest attack accuracies are averagely 3.54× higher than black-box for MS and 1.12× higher for MIA. Even worse, the highest attack accuracies (marked in red) are similar to that of the white-box baseline, indicating that the defense schemes are ineffective.

The relative accuracy (w.r.t. black-box baselines) of DarkneTZ (①) for MS is 3.92× and 1.28× for MIA. For Serdab (②), the relative attack accuracies are 4.03× and 1.34×. Since the attack performance toward both Serdab and DarkneTZ is high, we interpret that shielding a limited number of deep layers (①) or shallow layers (②) facilitates limited protection. Magnitude (③) achieves a similar defense effect with DarkneTZ, with 3.91× higher accuracy for MS and 1.25× higher accuracy for MIA. Though the DarkneTZ and Magnitude papers empirically demonstrate the defense effectiveness against naive adversaries (without the surrogate model initialized by a pre-trained model or public data), we depict that well-designed and practical attacks can crack such empirical settings.

SOTER (④) offers best protection across all solutions for MIA: in seven (out of 12) cases, SOTER achieves the lowest MIA accuracy among the five solutions. However, its higher security strength does not come for free. SOTER shields the largest number of layers using TEEs (20% layers) compared with other solutions. That is, SOTER has a much higher inference latency and utility cost.

Among the five schemes, ShadowNet (⑤) shows the weakest protection and most "red cases" in Table 2: five (out of 12) for MS and six for MIA. The average attack accuracy against ShadowNet is similar to that of the No-Shield baseline. According to our evaluation [5], the adversary can recover 95% of the obfuscated weights. This indicates that its lightweight obfuscation (matrix obfuscation and filter permutation) is insufficient to protect the target model in front of well-designed attacks.

> **Answer to RQ1**: Contrary to our expectation, existing TSDP solutions do not provide a black-box level security guarantee when being exploited by MS and MIA. That is, their shielded model weights and private training data are vulnerable to attackers with well-prepared M_{init} on hand.

4. Challenges of Straightforward Mitigations

Our study and observation in answering **RQ1** show that straightforward mitigation to the attacks for existing TSDP approaches is to put a larger proportion of a DNN model into TEEs to improve the protection effectiveness. However, this straightforward solution needs to address the *Security vs. Utility Trade-off*: putting more portions of a DNN model into TEEs boosts security but presumably diminishes utility (e.g., increases prediction latency). Thus, the objective is to find a "sweet spot" configuration (i.e., how large the TEE protected part should be) that satisfies the security requirement while minimizing the utility overhead. This section will therefore address the following research question:

> **RQ2:** For each of the five TSDP solutions evaluated in Sec. 3, is there a systematic approach to identify its "sweet spot" configuration that simultaneously achieves high utility and security?

4.1. Problem Formalization

To systematically find the "sweet spots" of the optimal size of the TEE shielded part for the TSDP solutions in

TABLE 2: Attack accuracies regarding representative defense schemes. "C10", "C100", "S10", and "UTK" represent CIFAR10, CIFAR100, STL10, and UTKFace, respectively. The last row reports the average accuracy toward each defense relative to the baseline black-box solutions. For each setting, we mark the highest attack accuracy in red and the lowest accuracy in yellow. Attack accuracy toward our approach (Sec. 5) is marked with green.

		Model Stealing ↓								Membership Inference ↓							
		No-Shield	①DarkneTZ	②Serdab	③Magnitude	④SOTER	⑤ShadowNet	Ours	Black-box	No-Shield	①DarkneTZ	②Serdab	③Magnitude	④SOTER	⑤ShadowNet	Ours	Black-box
AlexNet	C10	83.72%	77.15%	63.58%	65.97%	76.90%	83.57%	19.04%	24.38%	67.25%	57.67%	62.96%	52.67%	62.18%	69.43%	50.00%	50.00%
	C100	56.60%	41.57%	46.48%	47.86%	50.83%	56.43%	8.27%	10.68%	78.32%	63.27%	72.20%	71.31%	63.39%	81.23%	50.00%	50.00%
	S10	76.55%	75.17%	69.06%	73.67%	37.60%	35.98%	24.15%	15.26%	65.12%	58.49%	61.51%	66.26%	59.72%	65.57%	50.00%	50.00%
	UTK	90.01%	88.74%	82.92%	86.65%	58.86%	73.93%	52.27%	48.62%	62.97%	55.84%	55.43%	56.28%	55.52%	63.53%	50.00%	50.00%
ResNet18	C10	95.91%	87.55%	93.94%	89.92%	92.61%	91.58%	31.40%	19.88%	70.37%	65.01%	66.59%	59.12%	52.67%	69.53%	50.00%	50.00%
	C100	81.63%	70.11%	78.01%	74.84%	79.28%	78.51%	10.90%	15.41%	82.75%	81.10%	82.92%	67.55%	76.31%	83.73%	50.00%	50.00%
	S10	87.45%	86.03%	85.05%	77.08%	80.83%	84.38%	29.19%	21.66%	76.09%	65.98%	74.22%	64.29%	59.83%	74.07%	50.00%	50.00%
	UTK	90.78%	85.65%	84.65%	64.99%	76.43%	89.42%	51.95%	45.41%	62.87%	56.33%	59.25%	54.53%	51.69%	63.62%	50.00%	50.00%
VGG16_BN	C10	92.95%	87.76%	91.34%	87.35%	81.52%	90.67%	30.87%	14.62%	63.17%	64.03%	62.44%	58.63%	55.20%	62.14%	50.00%	50.00%
	C100	72.78%	63.68%	72.19%	68.82%	66.06%	72.85%	9.78%	10.93%	81.22%	78.63%	81.34%	71.25%	50.10%	81.13%	50.00%	50.00%
	S10	90.03%	89.17%	89.33%	84.33%	89.46%	89.43%	32.92%	18.97%	66.08%	68.20%	66.20%	66.97%	58.22%	65.85%	50.00%	50.00%
	UTK	91.51%	87.60%	89.60%	90.28%	87.30%	91.14%	48.37%	45.46%	58.73%	52.79%	58.48%	58.93%	51.34%	57.17%	50.00%	50.00%
Average		4.28×	3.92×	4.03×	3.91×	3.76×	4.28×	1.23×	1.00×	1.39×	1.28×	1.34×	1.25×	1.16×	1.39×	1.00×	1.00×

Section 3, we first formalize the problem as an optimization problem. Formally, let P be a TSDP solution that splits a DNN model into TEE-shielded and GPU-offloaded portions. Let C denote a configuration instance of P that specifies to what degree the model is shielded. We define two evaluation functions, $Security(C)$ and $Utility(C)$, which quantify the security risk and the utility cost of C, respectively. We also define $Security_{black}$ as the security risk baseline of a black-box setting, which puts the whole DNN model in TEE. As noted in Sec. 3.3, $Security_{black}$ denotes the lower bound of the security risk (the strongest protection TSDP can offer). Then, given the security requirement Δ, we formulate the "sweet spot" configuration C^* that satisfies $\mid Security(C) - Security_{black} \mid < \Delta$ with the minimal $Utility(C)$. Alternatively, the "sweet spot" is the solution to Equation 1.

$$C^* = \underset{\mid Security(C)-Security_{black} \mid <\Delta}{\arg\min} Utility(C) \quad (1)$$

4.2. Experimental Settings

To answer **RQ2**, we empirically identify the "sweet spots" for the five TSDP defenses evaluated in Sec. 3 w.r.t. different security risk metrics, datasets, and shielded models. We discuss the details of the experiment setup below.

Security Risk Metric. Consistent with Sec. 3.4, we implement $Security(C)$ using seven security metrics. For MS, we use model stealing accuracy, fidelity, and ASR. For MIA, we use confidence-based MIA accuracy, gradient-based MIA accuracy, generalization gap, and confidence gap. Following Sec. 3.3, we mainly report the results of MS accuracy (denoted as "Model Stealing") and confidence-based MIA accuracy (denoted as "Membership Inference").

Utility Cost Metric. As a common setup, we use FLOPs to measure the utility cost of DNN models [40]. FLOPs is a platform-irrelevant metric to assess the utility cost $Utility(C)$ by counting the total number of multiplication and addition operations conducted inside TEEs. We define $\%FLOPs$ as the ratio of FLOPs in the TEE over the total FLOPs of the DNN model. According to prior work [94], the computation speed inside TEE is about 30× slower than GPUs. Thus, a larger $\%FLOPs(C)$ indicates fewer computations are offloaded on GPUs, leading to higher utility costs.

We compute the FLOPs of different layers as follows. For a DNN layer, suppose the input channel size is c_{in}, the output channel size is c_{out}, and the width and height of the output are w and h. The FLOPs of a linear layer is computed as $2 \times c_{in} \times c_{out}$. The FLOPs of a batch normalization layer are computed as $2 \times c_{in} \times h \times w$. For a convolution layer, suppose the kernel size is k, the FLOPs are computed as $2 \times c_{in} \times k^2 \times h \times w \times c_{out}$. To validate the correctness of using $\%FLOPs$ as the utility measurement, we measured the inference latency of different TSDP solutions over different configurations on Intel SGX with an industrial-level platform [87]. Experimental results show that the inference latency increases monotonously with $\%FLOPs$; details in the Website [5].

Datasets and Models. The dataset and model selection is the same as Sec. 3.4. Due to space limitations, we will report the results on AlexNet, ResNet18, and VGG16_BN. The results of ResNet34 and VGG19_BN are displayed in the website [5].

Configurations. For each TSDP defense benchmarked in Sec. 3, we iterate possible configurations to identify C^*. In particular, for the defense that shields deep layers (①), we shield different numbers of consecutive "deep" layers starting from the output layer with TEEs. Similarly, for ②, which shields shallow layers, we put different amounts of consecutive layers starting from the DNN input layer. For ResNet models, we use the residual layers as the dividing boundaries. For VGG models and AlexNet models, we use convolution layers as boundaries.

For shielding large-magnitude weights (Magnitude; ③), the number of protected weights is controlled by a configuration parameter `mag_ratio`. We set the range of `mag_ratio` as $\{0, 0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$, whereas 0.01 is the recommended setting of Magnitude. For shielding intermediate layers (SOTER; ④), the number of

shielded layers is also defined by a configuration parameter, `soter_ratio`. We set the range of `soter_ratio` as $\{0, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1\}$ and 0.2 is the recommended setting of the original paper. For both ③ and ④, setting `mag_ratio` (and `soter_ratio`) to 0 represents the white-box baseline while setting the parameters to 1 is the black-box baseline. For shielding non-linear layers (ShadowNet; ⑤), we clarify that ShadowNet does not need to set any configuration.

Attack Implementation. We re-run the same MS and MIA as in Sec. 3.3. That is, we re-launch the three-phase attack pipeline, which comprises surrogate model initialization (P_i), model stealing (P_{ii}), and membership inference (P_{iii}).

4.3. Qualitative and Quantitative Results

We compute both qualitative and quantitative results to explore if the "sweet spot" configuration exists for each scheme and the characteristics of the sweet spots. For qualitative results, Fig. 3 presents the relationship between $Security$ and $Utility$ of different configurations. Note that in Fig. 3, we only show the results for MS accuracy ("Model Stealing") and confidence-based MIA accuracy ("Membership Inference") due to the space limit. In Fig. 3, the x-axis shows $\%FLOPs$ ($Utility$) of each partition configuration, and the y-axis shows the accuracy of MS and MIA ($Security$). For each model, MS and MIA are displayed in two sub-figures, respectively. In each sub-figure, shielding deep layers (①), shielding shallow layers (②), shielding large-magnitude weights (③), shielding intermediate layers (④), and shielding non-linear layers (⑤) are represented by a blue line with crosses, a green line with up triangles, an orange line with down triangles, a pink line with right triangles, and an aquamarine circle, respectively. We also plot the performance of the white-box baseline and the black-box baseline using horizontal black lines.

In a holistic sense, Fig. 3 suggests that there is no systematic and automated approach to identifying the "sweet spot" for the five representative defenses. The shapes of the lines are substantially different across datasets and models. Given a requirement Δ for security risk, it is tough to set a uniform threshold for $Utility(C^*)$ without a comprehensive empirical measurement of both $Security$ and $Utility$. For example, for shielding deep layers (①) of AlexNet for model stealing (the first row in Fig. 3), the shape of the curves for CIFAR10 and CIFAR100 are very different from the curve for STL10. Given a requirement Δ and a model to protect, the locations of "sweet spots" are random for different datasets.

For quantitative results, we measure the values of the $Utility(C^*)$ as defined in Equation 1 (the smallest value of $Utility$ to achieve $Security_{black}$) for different defenses. Table 3 reports the ratio of TEE-shielded FLOPs ($\%FLOPs$) to achieve $Security_{black}$ for each setting for MS and MIA. We omit the approach of shielding non-linear layers (ShadowNet, ⑤) because it does not require configurations. Overall, Table 3 implies that *the $Utility$ values to achieve $Security_{black}$ are distinct across protected models and datasets.* For example, to protect AlexNet from MS with shielding deep layers (①), we need to put 100% of the protected model in TEE to achieve $Security_{black}$ for CIFAR10 (C1) and CIFAR100 (C100). However, for STL10 (S10) and UTKFace (UTK), we only need to put 39.44% of FLOPs in TEE to achieve $Security_{black}$. Further, it is tough, if not impossible, to predict the actual value of $Utility(C^*)$ before we run the models empirically because the numbers are irregular. We also observe similar irregularity for other defense methods when protecting different models and datasets.

Answer to RQ2: It is difficult to systematically identify the "sweet spots" configuration C^* for prior TSDP solutions.

5. Design of TEESLICE

Besides systematically benchmarking de facto TSDP solutions (**RQ1**) and summarizing their common drawbacks (**RQ2**), we conclude the root cause of the TSDP solutions' vulnerabilities and propose a novel partition scheme, which alleviates the security vs. utility trade-off and can automatically find the "sweet spot" configuration.

The root cause of TSDP solutions' weakness is that all the TSDP approaches follow a *training-before-partition* strategy, which first trains M_{vic} using private data and then partitions the private model. After training, all the model weights (including the offloaded part) are updated by the private data and thus contain private information. During deployment, the private information in the offloaded weights is exposed to the untrusted environment. As demonstrated in **RQ1** and **RQ2**, attackers could effectively recover private information of M_{vic} from offloaded privacy-related weights with the help of public knowledge, *i.e.*, a well-prepared M_{init} on hand. With this regard, we champion that an ideal TSDP solution should ensure *the offloaded DNN weights on GPUs are never trained using private data* and thus, no information is leaked to the untrusted environment.

5.1. Approach Overview

We propose TEESLICE, a novel partitioning strategy that offloads DNN layers with no private information to GPUs at the inference stage. Prior solutions use public pre-trained model and private data to train M_{vic}, use heuristic designs to partition M_{vic}, and shield a subset of model parts. On the contrary, TEESLICE uses a *partition-before-training* paradigm, which partitions private data from the pre-trained model and then separately trains privacy-related layers. Fig. 4 shows the comparison between existing TSDP solutions (training-before-partition) and TEESLICE (partition-before-training).

We term the public pre-trained model as *backbone*, and the privacy-related layers as *model slices* [105], [107], [108], [106]. The model slices are lightweight and contain the private knowledge of M_{vic}. The backbone and model slices are combined to form a *hybrid model* (denoted as M_{hyb})

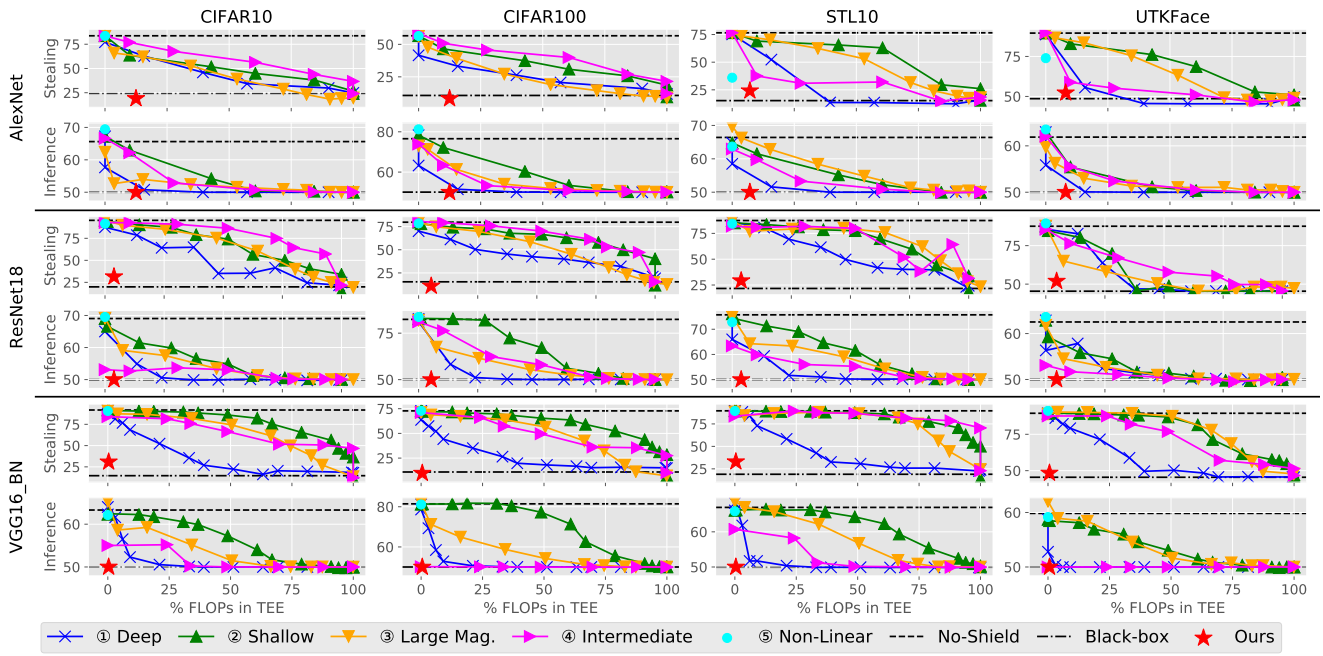

Figure 3: The relationship between $Security$ (y-axis) and $Utility$ (x-axis) of various TSDP solutions. The results of MS and MIA for each case are shown in two sub-figures. Each curve of " ①Deep", "②Shallow", "③Large Mag.", "④Intermediate", and "⑤Non-Linear" shows the corresponding solution. We also plot white-box and black-box baselines in horizontal lines.

TABLE 3: Different $Utility(C^*)$ ($\%FLOPs(C^*)$) values of "sweet spot" in front of MS and MIA. A lower value represents a lower utility cost. The $\%FLOPs(C^*)$ for white-box and black-box baselines are 0% and 100%, respectively. For each TSDP solution (row), we mark the lowest $Utility(C^*)$ with yellow and the highest value with red. For each case (model and dataset, column), we mark the lowest $Utility(C^*)$ across all solutions with green. The last column is the average utility cost for each solution. We omit shielding non-linear layers (ShadowNet, ⑤) because it does not require configurations.

		AlexNet				ResNet18				VGG16_BN				Average
		C10	C100	S10	UTK	C10	C100	S10	UTK	C10	C100	S10	UTK	
Model Stealing	①Deep	100.00%	100.00%	39.44%	39.44%	100.00%	100.00%	100.00%	37.46%	100.00%	100.00%	100.00%	69.23%	82.13%
	②Shallow	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	38.55%	100.00%	100.00%	100.00%	100.00%	94.88%
	③Large Mag.	81.18%	90.58%	100.00%	71.82%	100.00%	94.71%	100.00%	61.48%	100.00%	87.43%	100.00%	100.00%	90.60%
	④Intermediate	100.00%	100.00%	84.31%	60.69%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	95.42%
Membership Inference	①Deep	15.78%	15.78%	15.78%	15.78%	23.97%	23.97%	23.97%	23.97%	9.03%	21.07%	6.02%	3.01%	16.51%
	②Shallow	60.56%	84.22%	60.56%	42.81%	62.54%	86.52%	76.03%	38.55%	66.90%	90.97%	90.97%	60.88%	68.46%
	③Large Mag.	34.51%	53.17%	71.82%	34.51%	61.48%	61.48%	76.61%	44.93%	50.56%	66.47%	66.47%	50.56%	56.05%
	④Intermediate	60.69%	60.69%	60.69%	27.95%	72.80%	72.80%	72.80%	11.02%	33.84%	0.10%	33.84%	0.10%	42.28%
Ours		12.48%	12.48%	7.12%	8.01%	3.80%	5.33%	3.80%	4.58%	0.34%	0.47%	0.40%	0.60%	4.95%

that imitates the behavior of M_{vic}. Each slice takes the output of the prior layer (of the backbone) as its input and produces the input for the next layer (of the backbone). A detailed illustration of TSDP is shown in Fig. 1, where the yellow rounded blocks (Layer1 to Layer4) represent layers of the backbone and the blue squares (Slice1 and Slice2) are privacy-related model slices. The arrows between backbone layers and model slices indicate the data flow of internal DNN features. For example, the output of Layer1 is fed to Slice1, and Layer4 takes the outputs of Slice2 and Layer3 as input.

The key challenge for TEESLICE is to generate small slices that can run in TEEs with enough little or no accuracy lost. To this end, TEESLICE leverages a two-staged approach. First, it builds an instance of M_{hyb}, a densely sliced model (denoted as M_{dense}), with substantial private slices that can achieve high accuracy. However, M_{dense} cannot fit into TEEs due to its large size. Then, TEESLICE prunes M_{dense} with a self-adaptive, iterative pruning strategy that produces fewer slices without losing the M_{hyb}'s performance. The pruned model is another instance of M_{hyb}, which we call a sparsely sliced model (denoted as M_{sparse}). As the pruning is conducted simultaneously with the training phase, TEESLICE can prune slices with little accuracy drop. Note that the iterative pruning strategy is specifically designed for the resource-constrained TEE environment.

TEESLICE is partially motivated by NETTAILOR [69], which implements a training framework of M_{dense}. However, NETTAILOR does not aim to protect model privacy with TEEs and generates a large number of slices. To meet

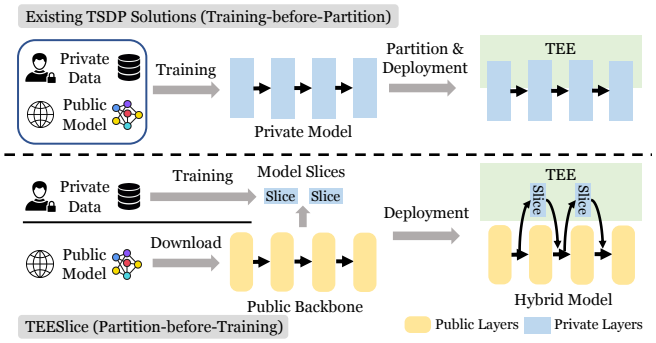

Figure 4: The comparison between TEESLICE and prior TSDP solutions. For TEESLICE, all information generated by private data will be handled in the TEE.

the constraint of TEEs, TEESLICE generates M_{sparse} with an adaptive pruning strategy to reduce the computation cost of slices. Besides, NETTAILOR lacks cryptographic primitives to securely transmit DNN intermediate data between GPUs and TEEs. TEESLICE employs a one-time pad (OTP) [11] and Freivalds' algorithm [31] to secure TEE-GPU transmission.

5.2. Detailed Design

TEESLICE consists of two stages: model slice extraction (training phase) and hybrid model deployment (inference phase). The slice extraction stage automatically finds the "sweet spot" by minimizing the utility cost while maintaining accuracy and security. TEESLICE trains M_{dense} from the public backbone and then prunes M_{dense} to get M_{sparse}. In the hybrid model deployment stage, M_{sparse} is deployed across the TEE and GPU. Model slices and non-linear layers (of the backbone) are deployed inside TEEs, whereas the other part of the backbone is offloaded on GPUs.

5.2.1. Model Slice Extraction. Densely Sliced Model Generation. Let the i-th layer of the public backbone be L_i. The private slice is represented as A_p^i, which connects layer L_p with layer L_i. A_p^i is designed to be lightweight and is $18\times$ smaller than L_i. The slices in M_{dense} connect a layer pair from backbone whenever the distance of the layers in this pair is less than three. During the training stage, TEESLICE assigns one importance scalar α_p^i to each slice A_p^i following the same strategy in related work [69]. The output of A_p^i is multiplied by α_p^i and sent to the next layer (of the backbone). A smaller α_p^i diminishes the influence of A_p^i. The model slices and the scalars are optimized simultaneously with a loss function that penalizes both model performance and complexity. The output of this step is M_{dense} with close performance as M_{vic}.

Iterative Slice Pruning. The pruning algorithm is guided by the importance scalar α_p^i, which controls the impact of A_p^i on the model prediction. We iteratively prune the slices with the smallest α_p^i and re-train the hybrid model to maintain the desired accuracy. A pre-defined threshold δ (defined as 1%) governs the tolerable accuracy loss during pruning. Let ACC_{vic} represent the accuracy of M_{vic}. The tolerable accuracy is $ACC_{\text{tol}} = (1-\delta) \cdot ACC_{\text{vic}}$, and the accuracy of the final M_{hyb} should be greater than ACC_{tol}.

Alg. 1 depicts the pruning pipeline. α_{setup} determines how to prune unimportant slices during the setup phase. The slice layers with $\alpha_p^l < \alpha_{\text{setup}}$ are pruned. As a heuristic, we set α_{init} to be 0.05 according to NETTAILOR. Iterative pruning requires two hyper-parameters: the number of the pruned slices in each round n and the total number of training rounds $rounds$. Each round begins with an evaluation of the model's accuracy ACC_r. If the current model satisfies the performance requirement ($ACC_r > ACC_{\text{tol}}$), TEESLICE prunes n model slices with the smallest α_p^l and trains the pruned model. Otherwise, TEESLICE skips the pruning operation and continues training the model.

Algorithm 1: Iterative Slice Pruning.

Input: The densely sliced model M_{dense}, pre-defined parameters α_{setup}, n, and $rounds$
Output: The hybrid model M_{hyb}
Prune M_{dense} by α_{setup} to get M_1 ;
for $r \leftarrow 1$ **to** $rounds$ **do**
  Compute the accuracy ACC_r of M_r ;
  if $ACC_r > ACC_{\text{tol}}$ **then**
    Store the model $M_{\text{hyb}} = M_r$;
    Select n slices with smallest α_p^l ;
    Prune the selected slices A_p^l ;
  end
  Re-train M_r to get M_{r+1} ;
end
return The hybrid model M_{hyb}

Automatically Find the Sweet Spot. The iterative slice pruning is indeed an optimization process that automatically finds the "sweet spot." Given the constraints of model security and accuracy, the iterative slice pruning optimizes the size of the private model slices to reduce the utility cost. Alg. 1 only explicitly considers the threshold ACC_{tol} for accuracy lost because all the private information is in the slices, which will run in TEEs. Therefore, the confidentiality of M_{vic} remains intact after offloading the backbone. Unlike prior TSDP solutions, where it is difficult to find the sweet spot configuration without a comprehensive evaluation of both security and utility (Sec. 4), TEESLICE does not have this shortcoming.

5.2.2. Hybrid Model Deployment. When deploying M_{hyb}, the private slices and non-linear layers (of the backbone) are shielded by the TEE, while the GPU hosts the backbone's linear layers. Shielding non-linear layers of the backbone is a common practice for prior TSDP solutions [94], [38], [74], [40] because non-linear layers are hard to securely offload to GPUs and only occupy a small fraction (about 1.5%) of the DNN's computation cost [94]. In the illustration figures (Fig. 1 and Fig. 4) we omit the non-linears in TEE for simplicity. There are two security challenges to deploy M_{hyb}: 1) how to encrypt features transmitted between GPU and TEE, and 2) how to verify the correctness of computations offloaded on GPUs. These two challenges can be

solved separately using one-time pad (OTP) and Freivalds' algorithm [31].

Feature Encryption. For a backbone linear layer $g(\cdot)$, let $\mathbf{h}$ be the plaintext input shielded by TEE. We first quantize $\mathbf{h}$ into a 8-bit representation following prior literature [94], [74] and get $\hat{\mathbf{h}}$. Then, we select a large prime value p, generate a random mask $\mathbf{r}$ (as the OTP), and encrypt the feature by

$$\mathbf{h}_e = (\hat{\mathbf{h}} + \mathbf{r}) \ \% \ p. \tag{2}$$

GPU receives $\mathbf{h}_e$, computes $g(\mathbf{h}_e)$, and returns the result back to TEE. TEESLICE decrypts the result by computing $g(\hat{\mathbf{h}}) = g(\mathbf{h}_e) - g(\mathbf{r})$ because

$$\begin{aligned} & g(\mathbf{h}_e) - g(\mathbf{r}) = g((\hat{\mathbf{h}} + \mathbf{r}) \ \% \ p) - g(\mathbf{r} \ \% \ p) \\ = \ & g((\hat{\mathbf{h}} + \mathbf{r}) \ \% \ p - \mathbf{r} \ \% \ p) = \ g((\hat{\mathbf{h}} + \mathbf{r} - \mathbf{r}) \ \% \ p) \\ = \ & g(\hat{\mathbf{h}} \ \% \ p) \ = \ g(\hat{\mathbf{h}}). \end{aligned} \tag{3}$$

The last equation holds as long as $p > 2^8$. Note that following prior work [94], computing $g(\mathbf{r})$ can be conducted by the model provider or inside TEE in an offline phase. Both strategies do not increase the overhead of online inference and do not impede its utility.

Result Verification. Freivalds' algorithm can periodically verify the computation results on GPUs on all linear layers. Let the weight of the linear layer $g(\cdot)$ be $\mathbf{W}$ and $g(\mathbf{h}) = \mathbf{h}^\mathsf{T}\mathbf{W}$, TEESLICE samples a random vector $\mathbf{s}$ that has the same shape as $g(\mathbf{h})$. TEESLICE then pre-computes $\tilde{\mathbf{s}} = \mathbf{W}\mathbf{s}$. The verification can be conducted by checking $g(\mathbf{h})^\mathsf{T}\mathbf{s} = \mathbf{h}^\mathsf{T}\tilde{\mathbf{s}}$.

6. Experiments

We implement TEESLICE with PyTorch 1.7 and we select ResNet18 as the public backbone following NETTAILOR. We have the flexibility to choose the backbone as commonly-used models without affecting the security guarantee [69]. We select ResNet18 because it is the default setting of NETTAILOR. As a fair setting, we set the training time for M_{dense} and M_{sparse} to half the time required to train M_{vic}, respectively. Hence, the overall training time for M_{sparse} and M_{vic} are equivalent. We apply TEESLICE to all the datasets and victim models in Sec. 3.4. The experiments aim to answer the following RQs:

RQ3: How does TEESLICE compare with representative defenses in Sec. 3.3 w.r.t. security and utility?
RQ4: Does TEESLICE sacrifice the accuracy of the original model?
RQ5: What is performance of TEESLICE on real-world devices? How much can TEESLICE speed up compared to the shielding-whole-model baseline?
RQ6: How is TEESLICE's scalability to NLP tasks.

6.1. Security Guarantee and Utility Cost

Security Guarantee. We follow the same experiment protocol in Section 3 to compare TEESLICE with five representative TSDP schemes. Specifically, for TEESLICE, we assume the attacker knows the architecture of M_{hyb} (default assumption), including which public backbone it uses and the structure of privacy-related model slices. The attack pipeline is the same as in Section 3.

Table 2 reports the results, with attack accuracy against our approach marked in green. The results are highly promising: in all cases, the attack accuracies are comparable with black-box protection and are better than the best of existing defenses (marked with yellow). For MS, the relative accuracy of TEESLICE compared to the black-box baseline is 1.24×, while the relative value of the best defense, SOTER (④), is 3.76×. For MIA, the attack accuracy of TEESLICE is similar to the black-box baseline (random guess). It is because all the feature communications are encrypted, and the TEE shields all the privacy-related slices.

Security Under Other Assumptions of M_{sur}. This section evaluates the security guarantee of TEESLICE with two different assumptions. The first assumption, *backbone-only*, is a weaker one that assumes the attacker only knows the public backbone of TEESLICE and does not know the slice information (slice positions and structures). The second assumption, *victim-knowing*, is a stronger assumption that assumes the attacker knows the structure of the original M_{vic}. Note that the victim-knowing is *not* a realistic assumption, and we only evaluate it to show the performance of TEESLICE under different settings. We also follow the evaluate protocol and attack pipeline in Sec. 3 for a fair experiment.

We show the MS accuracies of the two additional assumptions with the default assumption (knowing the structure of M_{hyb}) in Table 4. Note we omit MIA because the additional assumptions do not introduce new information of M_{vic}'s training data. Thus the results of MIA are the same as the default assumption. For each model and dataset Table 4, we mark the highest accuracy with red and the lowest accuracy with green. From Table 4, we can see that victim-knowing has lower accuracies than the other two assumptions (seven green cells and two red cells). The default assumption (Hybrid M_{hyb}) and backbone-only perform similarly (both have five red cells). We suspect that the reason for victim-knowing's low accuracy is that the M_{vic} in Table 4 has a smaller model capacity than the backbone. In other experiments, we found a M_{vic} with larger capacity (ResNet34; see our website [5]) has the highest MS accuracy for all datasets.

Security Under Other Assumptions of Data. In Sec. 3 and Sec. 4, we study a realistic adversary that has a small amount of data. Although our assumption on the adversary is realistic [82], [44], [103], we still study the security of TEESLICE with an ideal adversary who has a large amount of data to verify if TEESLICE ensures the security of DNN models under extreme conditions. We compare MS accuracies on various M_{sur} and a large range of queried data size between our approach and black-box protection. The goal is to study if our approach increases MS accuracy under the new assumption. The M_{sur} includes M_{hyb}, the backbone (*i.e.* , ResNet18; backbone-only), and all the M_{vic}

TABLE 4: Comparison of model stealing accuracy between different attack assumptions of M_{sur}.

		Hybrid M_{hyb}	Backbone	Victim M_{vic}
AlexNet	C10	19.04%	19.56%	23.71%
	C100	8.27%	14.48%	11.9%
	S10	24.15%	32.75%	17.14%
	UTK	52.27%	51.32%	47.0%
ResNet18	C10	31.4%	25.63%	17.33%
	C100	10.9%	18.33%	7.78%
	S10	29.19%	32.77%	32.86%
	UTK	51.95%	50.86%	51.63%
VGG16_BN	C10	30.87%	25.65%	20.69%
	C100	9.78%	18.44%	6.38%
	S10	32.92%	32.51%	31.75%
	UTK	48.37%	52.54%	51.04%

in Sec. 3.4 (victim-knowing). Following prior work [77], we set the queried data sizes as {50, 100, 300, 500, 1K, 3K, 5K, 10K, 15K, 20K, 25K, 30K}.

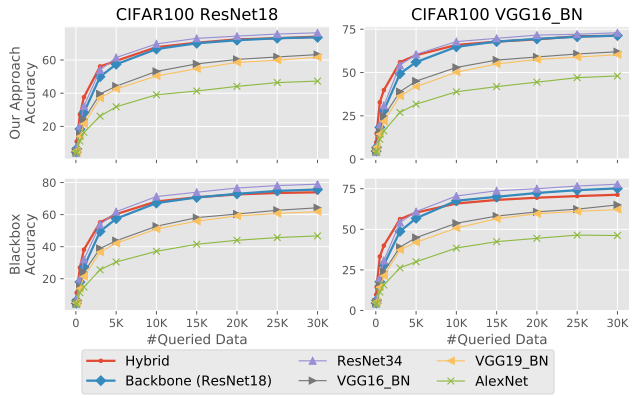

Figure 5: Comparison of TEESLICE and the black-box protection against MS attacks with different sizes of queried data. We report the accuracy of M_{sur}, where the first row represents TEESLICE and the second row is for the black-box baseline.

We display MS accuracy on CIFAR100 and two models (ResNet18 and VGG16_BN) in Fig. 5. The observation of other metrics and models is consistent with Fig. 5 and we put the other results on our website [5]. The first column of Fig. 5 means M_{vic} is a ResNet18 model and the second column means M_{vic} is a VGG16_BN. The first row of Fig. 5 displays the results of TEESLICE, and the second row shows the results of the black-box defense. We can observe from Fig. 5 that, for all cases, the MS accuracy against TEESLICE is similar to that of black-box baseline. According to the Wilcoxon signed-rank test [101], the null hypothesis is that there is no difference in accuracy. The p-value is 0.81, which cannot reject the null hypothesis. This result indicates that the differences between the accuracy of TEESLICE and the black-box baseline have no statistical significance. To summarize, under a different assumption of more queried data, the MS accuracy has no difference between TEESLICE and the black-box baseline.

Utility Cost. We qualitatively compare TEESLICE with the security-utility curves of other defenses in Fig. 3. For both MS and MIA, TEESLICE achieves a similar level of black-box defense with a distinguishably smaller $\%FLOPs$ than other defenses. ★, denoting TEESLICE, locates at the bottom left corners for all cases in Fig. 3. We also quantitatively compare the $Utility(C^*)$ of TEESLICE with other defenses in Table 3. For each case (column), we mark the lowest value of $Utility(C^*)$ with green. TEESLICE achieves the lowest utility cost in ten out of 12 cases. The average utility cost of TEESLICE is 4.95%. On the contrary, the average utility cost of other defenses ranges from 42.28% to 95.42%. That is, TEESLICE takes $10\times$ less utility cost to achieve the highest (black-box) defense level.

TABLE 5: The accuracy comparison between the victim model and the hybrid model trained by TEESLICE in the form of $M_{\text{vic}}/M_{\text{hyb}}$. Except for AlexNet where TEESLICE has a higher accuracy due to a larger model capacity, by average, TEESLICE's relative accuracy loss (the ratio between the accuracy of M_{hyb} and the accuracy of M_{vic}) is 0.34%.

	CIFAR10	CIFAR100	STL10	UTKFace
AlexNet	83.71%/86.37%	56.46%/61.96%	76.54%/80.17%	89.42%/88.92%
ResNet18	95.47%/93.65%	79.94%/76.79%	87.51%/86.22%	86.97%/88.24%
VGG16_BN	91.62%/93.06%	73.03%/73.11%	89.67%/89.42%	89.19%/89.46%

Answer to RQ3: TEESLICE features promising, black-box-level security guarantees under different attack assumptions. The utility cost ($Utility(C^*)$) of TEESLICE is $10\times$ less than other TSDP solutions.

6.2. Accuracy Loss

To answer this research question, we compare the accuracy between M_{vic} and their derived hybrid models M_{hyb} trained by TEESLICE. The result is in Table 5. In general, TEESLICE does not lead to a considerable loss of accuracy. For AlexNet, TEESLICE achieves a higher accuracy because the model capacity of the backbone (i.e., ResNet18) is larger than AlexNet. This phenomenon is consistent with the finding in Sec. 6.1 that AlexNet (the lowest model complexity) always has low accuracy. To statistically understand the accuracy loss, we compute the statistical significance using the Wilcoxon signed-rank test [101] across all models except AlexNet. The null hypothesis is that the accuracies between M_{vic} and M_{hyb} have no difference. The p-value is 0.75 and provides little statistical significance to reject the null hypothesis. Thus, the pruning processes have little effect on the accuracy of M_{hyb}.

We also measure the accuracy loss of eNNclave [84], a recent work sharing similar concepts, by putting M_{vic}'s last layer into TEE while replacing the GPU-offloaded shallow layers with a public backbone. As clarified in Sec. 3.1, eNNcalve suffers from low accuracy. We evaluate the accuracies of eNNclave over all models and datasets and find that eNNclave has an average downgrade of 34%, higher than (about $10\times$) our approach.

TABLE 6: The throughput comparison between shielding-whole-model, no-shield, and TEESLICE on a real desktop with SGX and GPU. We switch SGX to the hardware mode to enable all protections. In parentheses, we present the speedup w.r.t. the shielding-whole-model baseline.

	AlexNet	ResNet18	VGG16 BN
Black-box	6.56	7.67	1.55
No-Shield	495.27 (75.53×)	288.56 (36.56×)	103.10 (66.42×)
CIFAR10	44.67 (6.78×)	63.81 (8.32×)	72.80 (46.90×)
CIFAR100	47.36 (7.22×)	46.63 (6.08×)	58.69 (37.81×)
STL10	85.79 (13.08×)	65.24 (8.50×)	71.35 (45.97×)
UTKFaceRace	41.29 (6.30×)	58.03 (6.26×)	42.34 (27.28×)

Answer to RQ4: Besides achieving a principled security guarantee, TEESLICE doesn't undermine model accuracy.

6.3. Performance on Real-World Devices

We created a prototype framework on a Desktop PC with Intel Core i7-8700 3.20GHz CPU and NVIDIA GeForce GTX 1080 GPU to evaluate TEESLICE's speed-up on real devices. The framework has two parts: SGX's shielded section and the GPU's offloaded part. The SGX component is developed in C++ and is compiled using Intel SGX SDK 2.6 and GCC 7.5. The GPU component is built in PyTorch 1.7 and is supported by CUDA 11.7. We reused code from Goten [74] and Slalom [94] and implemented other TEESLICE's operations, such as convolution, the OTP-based feature encryption, and hybrid model architecture. We emulate production conditions by switching SGX to hardware mode with all its protection. We ran all experiments ten times and got the average inference time. We verify that the running time deviates less than 10% from the average. We mainly report the throughput (images per second) as it is more straightforward to evaluate the speed of ML systems [94]. The lowest required throughput for a real-time on-device ML service is 30 (a latency of 33ms) [16]. The throughput is computed by $1000/average\_latency$.

Table 6 presents the throughput of TEESLICE on three models (AlexNet, ResNet18, and VGG16_BN), as well as two baselines shielding-whole-model (in the SGX) and no-shield (on the GPU). Shielding-whole-model is the throughput lower bound, and no-shield is the upper bound. For each case, we display the speedup w.r.t. shielding-whole-model baseline in parentheses. For shielding-whole-model, the throughputs on the three models range from 1.55 to 7.67, far from the required throughput of real-time service (30). The throughput of no-shield baseline ranges from 103.10 to 495.27, much faster than the real-time requirement. Besides, the throughput speedup of no-shield compared with shielding-whole-model ranges from $36.56\times$ to $75.53\times$, demonstrating the efficiency of GPU. The throughputs of TEESLICE range from 41.29 to 85.79, which are, on average, $18.37\times$ faster than the shielding-whole-model baseline and satisfy the real-time requirement of ML services.

To further analyze the performance of TEESLICE, we also logged the latency of different parts during the inference phase. We break down the inference latency of TEESLICE into four parts: `Data Transfer`, `Slice in TEE`, `Backbone on GPU`, and `Non-Linear in TEE`. `Data Transfer` is the time to transfer internal results between SGX and GPU. `Slice in TEE` is the time to compute the private slices inside SGX. `Backbone on GPU` is the time to compute the convolution layers of the backbone on the GPU. `Non-Linear in TEE` is the time to compute the non-linear layers (*e.g.* ReLU) inside SGX (recall Sec. 5.2 that the ReLU layers of the backbone are computed inside TEE). Table 7 displays the percentage of each part over the total inference latency. From the table, we can see that `Slice in TEE` occupies 40.49% of the inference time due to the constrained computation resources inside SGX. `Data Transfer` and `Non-Linear in TEE` occupy 35.61% and 20.96% of the inference time because all the non-linear layers of the backbone are computed inside SGX. `Backbone on GPU` only occupies 2.84% of the time due to the strong computation ability of the GPU. Note that although TEESLICE introduces the additional overhead of `Data Transfer`, TEESLICE still accelerates the overall inference time by a large margin.

TABLE 7: TEESLICE inference time breakdown.

`Data Transfer`	`Slice in TEE`	`Backbone on GPU`	`Non-Linear in TEE`
35.61%	40.49%	2.84%	20.96%

TABLE 8: MS accuracy on NLP tasks against TEESLICE.

	SST-2	MRPC	RTE	Average
Black-box	50.92%	68.87%	48.38%	56.05%
No-Shield	92.55%	85.05%	66.79%	81.46%
TEESlice	50.92%	68.64%	46.93%	55.49%

Answer to RQ5: TEESLICE accelerates the throughput by an average of $18.37\times$ compared with the shielding-whole-model baseline and satisfies the real-time requirement.

6.4. Scalability to NLP Tasks

The design of TEESLICE is applicable to protect various DNN models. Thus, findings on protecting computer vision models in Sec. 6 are also applicable to NLP models. To demonstrate the generalization of TEESLICE, we evaluate TEESLICE on a representative NLP model, BART [58], and three NLP datasets (SST-2, MRPC, and RTE) from the popular GLUE dataset [98]. We mainly report MS accuracy in Table 8, and omit MIA accuracy as the partition strategy of TEESLICE does not leak additional membership information. The comparison baselines are "No-Shield" and "Black-box", aligned with Sec. 3. The results are generally consistent with Sec. 6. The attack accuracies of TEESLICE are comparable with "Black-box" and are lower than "No-Shield". Besides, the FLOPs of shielded layers by TEESLICE are significantly lower than "Black-box" (over $10\times$). Thus, we interpret that NLP models can also be effectively shielded by TEESLICE, and our findings in **RQ4** are generalizable to NLP models.

Answer to RQ6: TEESLICE is applicable to NLP tasks and manifests consistently high effectiveness.

7. Other Related Work

Besides the TSDP approaches in Section 3, we notice following areas related to securing DNN models with TEEs.

TEE in GPUs. Recent work explored implementing trusted architectures directly inside GPUs to achieve isolation [97], [43], [76]. Such solutions require customizing hardware and are designed for server centers. Our solution is primarily for user's end devices, which requires no change to the hardware or shipped firmware. Thus, this paper employs commercial GPUs in an "out-of-the-box" manner.

Side Channels. TEEs are known as vulnerable toward side-channel attacks [75], [17], [51], [20], [96], [70]. While side channels may threaten DNN privacy, various defensive methods have been proposed to mitigate side channel breaches [75], [55], [21], [35]. TEESLICE can be integrated with such defense to reduce side channel leakages.

Shielding-Whole-Model by TEE. We have reviewed existing TSDP solutions in Sec. 3.1. In addition to splitting DNNs and offloading certain parts of the model on GPUs to speedup model inference, we also notice existing works explore putting the entire DNN models into TEEs [56], [37], [59], [50], [87]. Nevertheless, these works often notably sacrifice the utility of the protected DNN models.

TSDP for DNN Training. Researchers have proposed various TSDP solutions for DNN training to protect the privacy of training data on the cloud server [38], [74], [94]. These solutions are different from TEESLICE because TEESLICE is designed to protect the model inference stage.

8. Discussion

Other Choices of $Security$ and $Utility$. This paper aims to comprehensively evaluate TSDP with seven empirical metrics of $Security$ from well-developed attack toolkits [61], [77]. These seven metrics cover the majority of MS and MIA in literature. We notice that differential privacy (DP) can also theoretically quantify $Security$ [29], but we decide not to use it due to its prohibitively high computational cost for large models. We leave the evaluation of other envisioned metrics in future work.

An intuitive choice of $Utility$ is the model inference latency. However, as there are various TEE architectures on the market, *e.g.*, Intel SGX [62], AMD SEV [49], Intel TDX [45], ARM CCA [13], and TrustZone [12], evaluating the latency on all DNN models, TEE architectures and possible configurations is difficult. We leverage FLOP, a platform-irrelevant function, to form $Utility$, and therefore, our conclusion should not be affected by the TEE implementation details, and is generally applicable to the wide range of TEE architectures.

Attacks to Black-box Models. Attackers can still compromise TEESLICE with black-box attacks [47], [77], [78], [81], [95], [60], [63]. However, the black-box attacks are *much less effective* due to the lack of information about model architectures and model weights. That is, we deem black-box attacks as the upper bound security guarantee that can be offered by TEEs. Several methods are proposed to mitigate black-box attacks [47], [78]; we view those defenses are orthogonal to TEE-based defenses.

Hyper-Parameters of TEESLICE. We clarify that although TEESLICE has involved hyper-parameters in the training phase, those parameters are merely used for reducing the computation cost inside TEE (amount of privacy-related slices) instead of influencing privacy leakage. The training phase of TEESLICE relies on several hyper-parameters, including δ, α_{setup}, n, and $rounds$. They are mundane in training our model setup. Thus, we clarify that it is unnecessary to tune those parameters and benchmark if their different values may influence privacy leakage.

Scalability to Large Language Models (LLMs). Over the last few months, LLMs (such as ChatGPT [4] and LLaMA [6]) have achieved great advances. The sizes of LLMs (containing up to hundreds of billions of parameters [3]) are larger than traditional CNNs (only hundreds of millions of parameters [7]) and thus introduce greater challenge to TEEs-shielded model protection solutions. However, we note that TEESLICE is also applicable to LLMs to protect the sensitive model privacy with TEEs. The partition-before-training strategy can be integrated with recent LLMs' parameter-efficient training techniques (*e.g.* LoRA [41]) to efficiently shield LLMs' critical privacy-related slices in the TEEs. The size of shielded slices is only a small fraction of the entire model (up to 10,000 times smaller [41]) and thus can significantly improve the inference latency. We believe TEESLICE can be a promising solution to protect the privacy of LLMs in the future.

New TEE Architectures. Desipte the traditional TEE architectures (*e.g.* Intel SGX [62] and ARM TrustZone [12]) that have been widely used in the industry, new TEE architectures are still emerging (*e.g.* Intel TDX [45] and ARM CCA [13]). Such new architectures may have stronger computation abilities. For example, Intel TDX has a larger encrypted memory of 1 TB [45]. Although such new TEEs may mitigate the performance overhead of shielding-whole-model solutions, they do not harm the practicality of TEESLICE because the computation speed of such new TEEs is still not comparable with GPUs, not to say the GPU architectures are also evolving. We believe TEESLICE can be a promising solution to bridge the gap between the new TEEs and the evolving GPUs.

Application Scope of TEESLICE. This paper focuses on an important application scope: protecting DNN privacy on the user's end devices with TEEs. With the development of hardware architectures, many mobile/IoT devices are already equipped with TEEs by default, such as TrustZone in Raspberry Pi [8] and Android 7 [1]. With the increasing awareness on the user privacy and companies' intellectual property embedded in the DNN model, we believe this topic will attract more attention in the future.

Threats to Validity. In Sec. 3 and Sec. 4, we use a practical adversary to evaluate existing solutions. The adversary has access to public data/models and constructs a surrogate

model to perform MS and MIA. We argue that this adversary is realistic and reasonable and we do not make any abnormal assumptions. The usage of public data/models is consistent with the assumption of existing works in this line of research [77], [99], [24], [78], [61].

Differential Privacy (DP). DP is a promising technique to theoretically quantify the privacy leakage of DNN training data to defense against MIA [29]. Nevertheless, DP is not designed to defense MS. Besides, recent works show that DP may provide insufficient privacy [67], harm utility/fairness [15], or degrade performance [91].

9. Conclusion

We have systematically examined existing TSDP solutions and uncovered their defects in front of privacy stealing attacks. Further, we illustrate the hurdles of identifying "sweet spot" DNN partition configurations, which frequently vary between models and datasets. With lessons harvested from attacking prior TSDP solutions, we present TEESLICE, a novel TSDP method that leverages the partition-before-training strategy. It achieves high-accuracy, high-security protection (comparable with shielding-whole-model baseline), and with much less (about $10\times$) computation overhead.

10. Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback of this paper. Ding Li and Yao Guo are corresponding authors. This work was partly supported by the National Natural Science Foundation of China (62172009,62141208). The HKUST authors were supported in part by the research fund provided by HSBC and the HKUST-VPRDO 30 for 30 Research Initiative Scheme under the contract Z1283.

References

[1] Android 7.0 Compatibility Definition. https://source.android.com/docs/compatibility/7.0/android-7.0-cdd#9.

[2] Artifact. https://github.com/ziqi-zhang/TEESlice-artifact.

[3] Awesome-LLM. https://github.com/Hannibal046/Awesome-LLM.

[4] ChatGPT. https://chat.openai.com.

[5] Full Supplementary. https://sites.google.com/view/tsdp-teeslice/home.

[6] Introducing LLaMA: A foundational, 65-billion-parameter large language model. https://ai.facebook.com/blog/large-language-model-llama-meta-ai/.

[7] Keras Applications. https://keras.io/api/applications/.

[8] OP-TEE documentation Raspberry Pi 3. https://optee.readthedocs.io/en/latest/building/devices/rpi3.html.

[9] Knockoff nets demo code. https://github.com/tribhuvanesh/knockoffnets, 2019.

[10] Ml-doctor demo code. https://github.com/liuyugeng/ML-Doctor, 2022.

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

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

[13] Arm. Introducing Arm Confidential Compute Architecture. https://developer.arm.com/documentation/den0125/0200/Arm-CCA-Extensions, 2021.

[14] A. Asvadishirehjini, M. Kantarcioglu, and B. A. Malin. GINN: fast GPU-TEE based integrity for neural network training. In A. Joshi, M. Fernández, and R. M. Verma, editors, *CODASPY '22: Twelveth ACM Conference on Data and Application Security and Privacy, Baltimore, MD, USA, April 24 - 27, 2022*, pages 4–15. ACM, 2022.

[15] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov. Differential privacy has disparate impact on model accuracy. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15453–15462, 2019.

[16] S. Bateni and C. Liu. Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems. In A. Gavrilovska and E. Zadok, editors, *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, pages 371–385. USENIX Association, 2020.

[17] J. V. Bulck, F. Piessens, and R. Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 178–195. ACM, 2018.

[18] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramèr. Membership inference attacks from first principles. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1897–1914. IEEE, 2022.

[19] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In N. Heninger and P. Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 267–284. USENIX Association, 2019.

[20] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. Lai. Sgxpectre: Stealing intel secrets from SGX enclaves via speculative execution. *IEEE Secur. Priv.*, 18(3):28–37, 2020.

[21] G. Chen, W. Wang, T. Chen, S. Chen, Y. Zhang, X. Wang, T. Lai, and D. Lin. Racing in hyperspace: Closing hyper-threading side channels on SGX with contrived data races. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 178–194. IEEE Computer Society, 2018.

[22] J. Chen, J. Wang, T. Peng, Y. Sun, P. Cheng, S. Ji, X. Ma, B. Li, and D. Song. Copy, right? A testing framework for copyright protection of deep learning models. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 824–841. IEEE, 2022.

[23] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang. When machine unlearning jeopardizes privacy. In Y. Kim, J. Kim, G. Vigna, and E. Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 896–911. ACM, 2021.

[24] Y. Chen, C. Shen, C. Wang, and Y. Zhang. Teacher model fingerprinting attacks against transfer learning. In K. R. B. Butler and K. Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3593–3610. USENIX Association, 2022.

[25] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In G. J. Gordon, D. B. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 215–223. JMLR.org, 2011.

[26] P. W. Code. Image Classification on STL-10. https://paperswithcode.com/paper/hybridnet-classification-and-reconstruction, 2018.

[27] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[28] Z. Deng, K. Chen, G. Meng, X. Zhang, K. Xu, and Y. Cheng. Understanding real-world threats to deep learning models in android apps. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 785–799. ACM, 2022.

[29] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 2014.

[30] T. Elgamal and K. Nahrstedt. Serdab: An iot framework for partitioning neural networks computation across multiple enclaves. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020, Melbourne, Australia, May 11-14, 2020*, pages 519–528. IEEE, 2020.

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

[32] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[33] Google. Tensorflow hub. https://www.tensorflow.org/hub, 2020.

[34] Google. Tensorflow transfer learning api. https://www.tensorflow.org/tutorials/images/transfer_learning, 2020.

[35] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 217–233. USENIX Association, 2017.

[36] Z. Gu, H. Huang, J. Zhang, D. Su, H. Jamjoom, A. Lamba, D. Pendarakis, and I. Molloy. Confidential inference via ternary model partitioning. *arXiv:1807.00969*, 2018.

[37] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021, virtual, June 19-25, 2021*, pages 3300–3309. Computer Vision Foundation / IEEE, 2021.

[38] H. Hashemi, Y. Wang, and M. Annavaram. Darknight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*, pages 212–224. ACM, 2021.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[40] J. Hou, H. Liu, Y. Liu, Y. Wang, P. Wan, and X. Li. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Trans. Dependable Secur. Comput.*, 19(6):4270–4284, 2022.

[41] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[42] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang. Membership inference attacks on machine learning: A survey. *ACM Comput. Surv.*, 54(11s):235:1–235:37, 2022.

[43] W. Hua, M. Umar, Z. Zhang, and G. E. Suh. Guardnn: Secure DNN accelerator for privacy-preserving deep learning. *CoRR*, abs/2008.11632, 2020.

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

[45] Intel. Intel Architecture Memory Encryp- tion Technologies Specification. https://software.intel.com/content/dam/develop/external/us/en/documents-tps/multi-key-total-memory-encryption-spec.pdf, 2021.

[46] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In S. Capkun and F. Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1345–1362. USENIX Association, 2020.

[47] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. PRADA: protecting against DNN model stealing attacks. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 512–527. IEEE, 2019.

[48] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In W. Enck and A. P. Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018.

[49] D. Kaplan, J. Powell, and T. Woller. Amd memory encryption. *White paper*, 2016.

[50] K. Kim, C. H. Kim, J. J. Rhee, X. Yu, H. Chen, D. J. Tian, and B. Lee. Vessels: efficient and scalable deep learning prediction on trusted processors. In R. Fonseca, C. Delimitrou, and B. C. Ooi, editors, *SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, October 19-21, 2020*, pages 462–476. ACM, 2020.

[51] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

[52] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[54] kuangliu. Train CIFAR10 with PyTorch. https://github.com/kuangliu/pytorch-cifar, 2020.

[55] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 557–574. USENIX Association, 2017.

[56] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In S. A. Brewster, G. Fitzpatrick, A. L. Cox, and V. Kostakos, editors, *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, Los Cabos, Mexico, October 21-25, 2019*, pages 46:1–46:17. ACM, 2019.

[57] K. Leino and M. Fredrikson. Stolen memories: Leveraging model memorization for calibrated white-box membership inference. In S. Capkun and F. Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1605–1622. USENIX Association, 2020.

[58] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, 2020.

[59] Y. Li, D. Zeng, L. Gu, Q. Chen, S. Guo, A. Y. Zomaya, and M. Guo. Lasagna: Accelerating secure deep learning inference in sgx-enabled edge cloud. In C. Curino, G. Koutrika, and R. Netravali, editors, *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, pages 533–545. ACM, 2021.

[60] Z. Li and Y. Zhang. Membership leakage in label-only exposures. In Y. Kim, J. Kim, G. Vigna, and E. Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 880–895. ACM, 2021.

[61] Y. Liu, R. Wen, X. He, A. Salem, Z. Zhang, M. Backes, E. D. Cristofaro, M. Fritz, and Y. Zhang. Ml-doctor: Holistic risk assessment of inference attacks against machine learning models. In K. R. B. Butler and K. Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 4525–4542. USENIX Association, 2022.

[62] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In R. B. Lee and W. Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013.

[63] S. Mehnaz, S. V. Dibbo, E. Kabir, N. Li, and E. Bertino. Are your sensitive attributes private? novel model inversion attribute inference attacks on classification models. In K. R. B. Butler and K. Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 4579–4596. USENIX Association, 2022.

[64] Meta. Pytorch hub. https://pytorch.org/hub/, 2020.

[65] Meta. Pytorch model zoo. https://pytorch.org/serve/model_zoo.html, 2020.

[66] F. Mireshghallah, M. Taram, P. Ramrakhyani, A. Jalali, D. M. Tullsen, and H. Esmaeilzadeh. Shredder: Learning noise distributions to protect inference privacy. In J. R. Larus, L. Ceze, and K. Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 3–18. ACM, 2020.

[67] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis. PPFL: privacy-preserving federated learning with trusted execution environments. In S. Banerjee, L. Mottola, and X. Zhou, editors, *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA, 24 June - 2 July, 2021*, pages 94–108. ACM, 2021.

[68] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In E. de Lara, I. Mohomed, J. Nieh, and E. M. Belding, editors, *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 161–174. ACM, 2020.

[69] P. Morgado and N. Vasconcelos. Nettailor: Tuning the architecture, not just the weights. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 3044–3054. Computer Vision Foundation / IEEE, 2019.

[70] K. Murdock, D. F. Oswald, F. D. Garcia, J. V. Bulck, D. Gruss, and F. Piessens. Plundervolt: Software-based fault injection attacks against intel SGX. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1466–1482. IEEE, 2020.

[71] K. G. Narra, Z. Lin, Y. Wang, K. Balasubramaniam, and M. Annavaram. Privacy-preserving inference in machine learning services using trusted execution environments. *CoRR*, abs/1912.03485, 2019.

[72] M. Nasr, R. Shokri, and A. Houmansadr. Machine learning with membership privacy using adversarial regularization. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 634–646. ACM, 2018.

[73] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 739–753. IEEE, 2019.

[74] L. K. L. Ng, S. S. M. Chow, A. P. Y. Woo, D. P. H. Wong, and Y. Zhao. Goten: Gpu-outsourcing trusted execution of neural network training. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 14876–14883. AAAI Press, 2021.

[75] A. Nilsson, P. N. Bideh, and J. Brorsson. A survey of published attacks on intel SGX. *CoRR*, abs/2006.13598, 2020.

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

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

[78] T. Orekondy, B. Schiele, and M. Fritz. Prediction poisoning: Towards defenses against DNN model stealing attacks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[79] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. K. Shevade, and V. Ganapathy. A framework for the extraction of deep neural networks by leveraging public data. *CoRR*, 2019.

[80] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[81] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In R. Karri, O. Sinanoglu, A. Sadeghi, and X. Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017.

[82] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1157–1174. IEEE, 2022.

[83] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[84] A. Schlögl and R. Böhme. ennclave: Offline inference with model confidentiality. In J. Ligatti and X. Ou, editors, *AISec@CCS 2020: Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security, Virtual Event, USA, 13 November 2020*, pages 93–104. ACM, 2020.

[85] T. Shen, J. Qi, J. Jiang, X. Wang, S. Wen, X. Chen, S. Zhao, S. Wang, L. Chen, X. Luo, F. Zhang, and H. Cui. SOTER: guarding black-box inference for general neural networks at the edge. In J. Schindler and N. Zilberman, editors, *2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 723–738. USENIX Association, 2022.

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

[87] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel SGX. In J. R. Larus, L. Ceze, and K. Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 955–970. ACM, 2020.

[88] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017.

[89] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[90] C. Song and A. Raghunathan. Information leakage in embedding models. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 377–390. ACM, 2020.

[91] P. Subramani, N. Vadivelu, and G. Kamath. Enabling fast differentially private SGD via just-in-time compilation and vectorization. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 26409–26421, 2021.

[92] Z. Sun, R. Sun, L. Lu, and S. Jha. Shadownet: A secure and efficient system for on-device model inference. *CoRR*, abs/2011.05905, 2020.

[93] Z. Sun, R. Sun, L. Lu, and A. Mislove. Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps. In M. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1955–1972. USENIX Association, 2021.

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

[95] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016.

[96] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida. RIDL: rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 88–105. IEEE, 2019.

[97] S. Volos, K. Vaswani, and R. Bruno. Graviton: Trusted execution environments on gpus. In A. C. Arpaci-Dusseau and G. Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 681–696. USENIX Association, 2018.

[98] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv:1804.07461*.

[99] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao. With great training comes great vulnerability: Practical attacks against transfer learning. In W. Enck and A. P. Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1281–1297. USENIX Association, 2018.

[100] weiaicunzai. Pytorch-cifar100. https://github.com/weiaicunzai/pytorch-cifar100, 2020.

[101] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. Springer, 1992.

[102] Y. Xiang, Y. Wang, H. Choi, M. Karimi, and H. Kim. Aegisdnn: Dependable and timely execution of DNN tasks with SGX. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*, pages 68–81. IEEE, 2021.

[103] M. Yan, C. W. Fletcher, and J. Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In S. Capkun and F. Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2003–2020. USENIX Association, 2020.

[104] X. Yuan and L. Zhang. Membership inference attacks and defenses in neural network pruning. *CoRR*, 2022.

[105] Z. Zhang, Y. Li, Y. Guo, X. Chen, and Y. Liu. Dynamic slicing for deep neural networks. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

[106] Z. Zhang, Y. Li, B. Liu, Y. Cai, D. Li, Y. Guo, and X. Chen. Fedslice: Protecting federated learning models from malicious participants with model slicing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 460–472. IEEE, 2023.

[107] Z. Zhang, Y. Li, J. Wang, B. Liu, D. Li, Y. Guo, X. Chen, and Y. Liu. Remos: Reducing defect inheritance in transfer learning via relevant model slicing. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*.

[108] Z. Zhang, L. K. Ng, B. Liu, Y. Cai, D. Li, Y. Guo, and X. Chen. Teeslice: slicing dnn models for secure and efficient deployment. In *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, pages 1–8, 2022.

[109] Z. Zhang, Y. Song, and H. Qi. Age progression/regression by conditional adversarial autoencoder. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4352–4360. IEEE Computer Society, 2017.

[110] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu. Hermes attack: Steal DNN models with lossless inference accuracy. In M. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1973–1988. USENIX Association, 2021.

[111] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proc. IEEE*, 109(1):43–76, 2021.

Appendix A.
Meta-Review

A.1. Summary

This paper aims to mitigate the model-stealing and membership-inference attacks against TEE-shielded DNN models. The authors study the weaknesses of existing solutions and, based on their observations, propose a "partition-before-training" strategy that partitions private data from the pre-trained model and then separately trains privacy-related layers. The evaluation results show that their new solution improves privacy with little accuracy loss.

A.2. Scientific Contributions

- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) This paper identifies an impactful vulnerability. Specifically, the authors demonstrate via a systematic evaluation that existing solutions that leverage TEEs to protect DNN model data and weights still have exploitable shortcomings when it comes to protecting privacy.
2) This paper provides a valuable step forward in an establish field by contributing a comprehensive and rigorous assessment of the current state-of-the-art techniques under various settings and criteria. They have selected two well-established attacks that target on-device ML models and demonstrated how the current techniques perform in mitigating them. They have also extended the assessment to identify an optimal setting for the current techniques that yields the best outcome, and shown that it is not a simple task to find a universal optimal setting.

A.4. Noteworthy Concerns

Several reviewers are concerned about the security and practicality of using OTP in the proposed solution. There has been extensive technical discussion regarding the security of OTP-based feature encryption schemes, starting with Slalom published in ICLR 2019. In a similar vein, reviewers expressed concerns about ensuring that the TEE does not exhaust the OTP during heavy long-term use, which might introduce an overhead not captured in the paper's evaluation. However, since OTP-based feature encryption schemes have been used in several prior solutions and has not been decisively proven insecure or impractical, the reviewers concluded that there is value in publishing this technique.

Appendix B.
Response to the Meta-Review

The meta-review notes the security and practicality of using OTP in TEESlice. For the security concern, the generated mask is never reused and the ciphertext lies in the finite field $\mathbb{Z}_p$ of integers modulo a prime p, thus it is provably secure against any cryptanalysis [94]. For the practicallity concern, we note that the size of random pads does not need to be long or infinite because we can send a small amount of pads at the setup phase and periodically update the pads. A straightforward solution is that TEESlice receives a proporate number of pads at the setup phase. When the pads are about to run up (e.g. less than 5%), TEESlice can request the model vendor for a batch of new pads. The OTP-based scheme has been adopted by prior work [94], [40], [92], thus we believe OTP will not harm the security and utility of TEESlice.