TBNet: A Neural Architectural Defense Framework Facilitating DNN Model Protection in Trusted Execution Environments

Ziyu Liu[1], Tong Zhou[1], Yukui Luo[2], Xiaolin Xu[1]
{liu.ziyu4,zhou.tong1,x.xu}@northeastern.edu,yluo2@umassd.edu
[1]Northeastern University, [2]University of Massachusetts Dartmouth

ABSTRACT

Trusted Execution Environments (TEEs) have become a promising solution to secure DNN models on edge devices. However, the existing solutions either provide inadequate protection or introduce large performance overhead. Taking both security and performance into consideration, this paper presents TBNet, a TEE-based defense framework that protects DNN model from a neural architectural perspective. Specifically, TBNet generates a novel Two-Branch substitution model, to respectively exploit (1) the computational resources in the untrusted Rich Execution Environment (REE) for latency reduction and (2) the physically-isolated TEE for model protection. Experimental results on a Raspberry Pi across diverse DNN model architectures and datasets demonstrate that TBNet achieves efficient model protection at a low cost.

ACM Reference Format:
Ziyu Liu[1], Tong Zhou[1], Yukui Luo[2], Xiaolin Xu[1]. 2024. TBNet: A Neural Architectural Defense Framework Facilitating DNN Model Protection in Trusted Execution Environments. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3658251

1 INTRODUCTION

Deep neural network (DNN) models are widely deployed on edge devices for diverse applications [14]. These on-device DNNs benefit from local model execution to reduce inference time. Moreover, on-device DNN mitigates the risk of data leakage, as sensitive data predominantly resides on the edge [4]. Despite these advantageous aspects of deploying DNNs on edge, it introduces security concerns. These well-trained models represent valuable intellectual property (IP), and the edge deployments expose them to model stealing attacks, in which adversaries can extract the model architecture and weights, using illegal memory access [17], side-channel attacks [1], or direct interactions with the model [5]. These attacks pose an emerging threat and may result in potential financial losses for model vendors.

Recently, the Trusted Execution Environments (TEEs) (e.g., ARM TrustZone [7]) are employed to secure edge DNN against model stealing attacks [18]. Although TEE ensures the confidentiality of DNN model and computation, its limited memory and computational resources, compared to the untrusted Rich Execution Environment (REE), make it inefficient to deploy the entire model within TEE [2]. There have been attempts to alleviate the storage and computation burden inside TEE while ensuring DNN model protection [10, 12]. These approaches, however, expose certain either well-trained layers or valuable DNN architectures to insecure memory, which can be possibly used by attackers to derive high-performance models, e.g., by fine-tuning valuable DNN architecture with enough data.

Generally, a TEE-base defense should achieve both secure and efficient deployment. Specifically, such a defense method should satisfy four critical requirements: **(1) preventing attackers from training/ fine-tuning the exposed partial models in REE to achieve a model with comparable accuracy to the victim model** and **(2) preventing attackers from reverse-engineering deployed layers in TEE** (e.g., infer architecture/computation inside TEE based on exposed information in REE). Simultaneously, the deployed model should **(3) preserve the accuracy of the victim model** and **(4) possess optimized computation and storage in TEE for efficient deployment.**

To meet these requirements, we propose TBNet, a novel architectural defense framework that generates a Two-Branch substitution model for secure deployment. In particular, TBNet only includes layer-wise connections from REE to TEE, but not vice versa. In this way, TBNet not only secures the critical architectural model confidentiality in TEE, but also enjoys the rich computation resources in REE for preserving performance. To achieve these goals, TBNet transfers partial knowledge from a well-trained victim model into the branch deployed in TEE (i.e., the secure branch). Meanwhile, the branch with partial knowledge remains in REE (i.e., the unsecured branch) to compensate for knowledge lost in the secure branch, preserving the accuracy of victim models. TBNet employs iterative two-branch pruning to protect the architecture of the victim model and further ease the storage and computation burden of TEE. To prevent the leakage of architectural model confidentiality in TEE through the unsecured branch, TBNet generates architectural difference between the secure and unsecured branches through rollback finalization. We make the following contributions in this paper:

- We propose *TBNet*, a TEE-based defense framework for DNN model. TBNet generates a novel two-branch substitution for secure and efficient deployment of victim models. This architecture defends against model stealing by preventing reverse-engineering of TEE layers and blocking attempts to train exposed partial models in REE to match the accuracy of the victim model.
- We employ knowledge transfer while optimizing storage and computation overhead in TEE, to preserve the performance of the victim model. By strategically distributing knowledge between two branches, our method achieves a balance between maintaining model accuracy and efficiently utilizing TEE resources.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3658251

- We evaluate TBNet on a Raspberry Pi across different model architectures and datasets. The experimental results demonstrate that TBNet significantly reduces TEE memory usage by up to 2.45× and lowers inference latency by up to 1.22× compared to fully executing victim models in TEE.

2 BACKGROUND AND RELATED WORKS

2.1 Trusted Execution Environment

A Trusted Execution Environment (TEE) is a hardware enclave that guarantees the confidentiality and integrity of data and computations inside [13]. It is accomplished through the isolation of this secure area on the main processor, employing a combination of hardware and software methods to ensure its protection. TEE possesses its own dedicated memory area and resources, functioning independently from the standard operating system. This design helps guarantee that untrusted users or applications are unable to access the content of the TEE. Standard operations occur in REE, while security-sensitive operations are reserved for TEE. Although TEE is promising in DNN model protection, it is inherently constrained by the memory and computational resources of edge devices that further impact the efficiency.

2.2 Threat Model

Without loss of generality, we adopt the threat model in related works [1, 5, 10], and elaborate its definition from the attacker and defender perspectives. **Attackers.** Their goal is to obtain a model with comparable performance to the victim model. An attacker can use different attacking methods [1, 5], e.g., illegal memory access [17], to extract the model architecture and weights in REE. Following the previous work [12], we consider TEE as a secure area and the data, code, and computation within it remain as a black box for attackers. We assume a strong attacker who can extract everything from REE, such as model parameters, computation, and even data transfer activities. Practically, we assume the original victim model from the model vendor is highly optimized. **Defenders.** They aim to ensure the confidentiality of DNN models while maintaining their performance. Specifically, the deployed model should perform well for users but prevent attackers from stealing. To achieve the performance-security balance, defenders can leverage TEE that is widely available on edge devices, such as ARM TrustZone [7].

2.3 Limitations of Prior Arts

TEE has been envisioned as a promising solution for DNN model protection, attributed to its secure memory and execution environment [12, 16, 18]. Prior works (e.g., [18]) proposed to partition the model and execute the inference sequentially to allow the model to be fully executed inside TEE, thus achieving full protection against model stealing. The drawback of this approach is that the overall inference latency is increased, because of the limited computation resources inside TEE. As an improvement, DarkneTZ [12] employs a strategy of partitioning model layers, only executing some sensitive layers within TEE and the remaining layers in the REE. However, the protection of model confidentiality is compromised by the exposure of certain convolution layers to attackers in plaintext format, as it reduces the effort and workload for the retraining process [19]. At the same time, the intermediate results transmitted between REE and TEE raise concerns. Specifically, the exposed well-trained layers produce detailed feature maps, serving as the input for the sensitive layers within the TEE. The feature maps generated by these TEE layers are then transmitted back to REE and decrypted in plaintext to execute. By closely monitoring both the input and output of these sensitive layers within the REE, attackers can systematically build and train substitute layers to replace the corresponding layers in the TEE with similar functionality, finally achieving model stealing. ShadowNet [16] transforms the weights of linear layers and outsources the intensive computation to untrusted hardware accelerators, subsequently restoring the result inside TEE. However, such a linear transformation approach is vulnerable to strong attackers, who can continuously monitor memory access patterns, match pairs of linear transformations and thereby recover the weights [20]. A recent work MirrorNet [10] proposes to execute a copy of the victim model in TEE to protect its confidentiality, which employs one-way communication to prevent the leakage of the computation results inside TEE. In this way, the data transmission turns out to be more secure, i.e., only being sent from REE to TEE. However, this solution fails to protect the original DNN architecture, which also represents a valuable model IP [11].

3 PROPOSED FRAMEWORK: TBNET

3.1 Design Requirements

From analyzing the limits of existing works, as well as the generic challenges of using TEE for DNN model protection, we summarize the following design requirements for a TEE-based defense solution:

- **Performance-preserving and optimization in TEE**. As a generic requirement for any security solutions, a defense framework should preserve the performance (e.g., inference accuracy) of a victim DNN model. Also, a defense framework can employ both the rich computational resources in REE and the physically secured TEE, to jointly conduct the DNN model inference. Considering the stringent resources in TEE, an efficient defense solution should optimize the model in TEE by reducing its memory usage and computation workload for better hardware efficiency.
- **One-way context switch between REE and TEE**. While utilizing TEE and REE together to execute a DNN model, a defense solution should address bidirectional intermediate data (e.g., feature map) transmission. Specifically, a defense solution should enforce the one-way context switch from REE to TEE, to prevent the possible model reconstruction using such data leakage.
- **Reduced model confidentiality exposure**. The sub-model executed in REE can leverage rich storage and computational resources, but also leak confidentiality (e.g., its architecture and weights). Therefore, a robust defense strategy must aim to thwart attackers from directly utilizing or fine-tuning the exposed sub-model to attain comparable performance to the victim model. Additionally, it should prevent attackers from deducing the architecture within TEE based on the sub-model in the REE, safeguarding the integrity and confidentiality of the deployed model.

To satisfy these design requirements, we propose TBNet, an architectural defense framework facilitating DNN model protection in TEE, by generating a novel Two-Branch model substitution. In particular, the newly generated model has a lightweight portion within TEE and can achieve similar performance to the victim

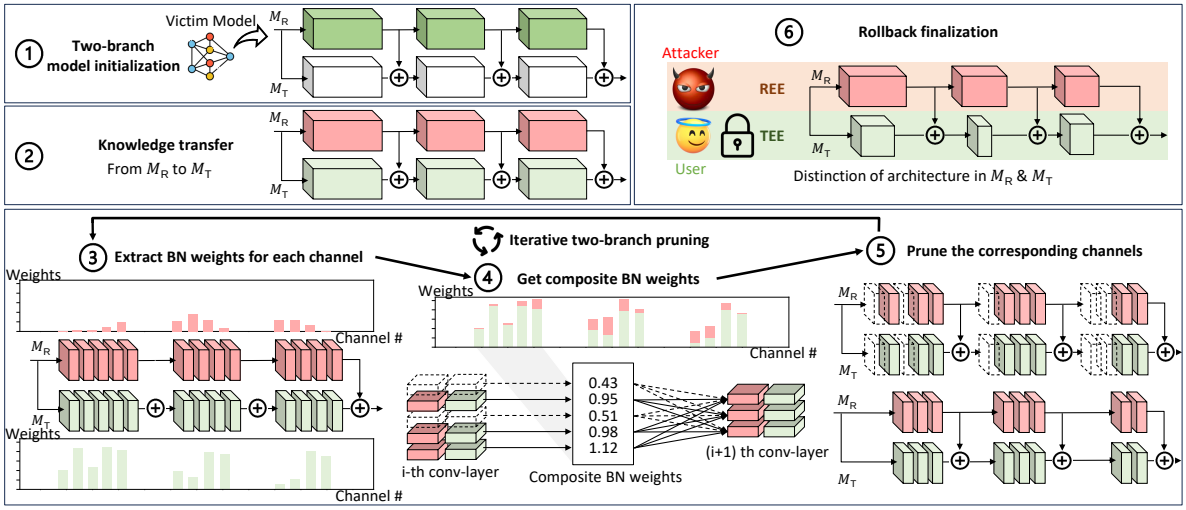

Figure 1: Workflow of the framework. Step ①: TBNet takes the victim model as the unsecured branch (M_R), then initializes a secure branch (M_T) that has the same model architecture as M_R. Step ② transfers the knowledge of the victim model (i,e., now M_R) to M_T. Step ③-⑤ apply iterative two-branch pruning to hide the architecture of the victim model and to obtain a lightweight M_T to ease storage and computational burden in TEE. Step ⑥ employs rollback to introduce an architectural distinction between M_R and M_T.

model. Fig. 1 illustrates the workflow of TBNet, which includes 6 steps, detailed as follows.

3.2 Step ①: Two-branch Model Initialization

We enforce one-way data transmission to avoid leaking the intermediate feature maps generated in TEE. In order to achieve this, we initialize a two-branch model by taking the victim model as the unsecured branch executed in REE (denoted by M_R), then constructs the secure branch executed in TEE (denoted by M_T) with the exact same architecture as the M_R. The reason we employ the same architecture in two branches is to incorporate the feature map result from M_R into M_T. Particularly, the feature maps of one layer computed by M_R will be transmitted from REE to TEE and element-wise added with the feature maps generated from M_T, which serves as the input of the subsequent layer of M_T (see ① in Fig. 1).

With such an architectural design to enhance the confidentiality of computation in TEE, it is also crucial to prevent attackers from directly leveraging the M_R to achieve satisfactory accuracy, thereby minimizing the risk of model stealing. As the current M_R is initialized with knowledge equivalent to the victim model, our subsequent objective is to transfer this knowledge from the unsecured branch to the secure branch.

3.3 Step ②: Knowledge Transfer

To reduce the model confidentiality exposed in REE, TBNet aims to transfer partial knowledge from M_R to M_T. Specifically, M_R inherits the model architecture and weights from the original victim model, encapsulating valuable "knowledge", in terms of discerning feature characteristics and accurate categorization.

This knowledge transfer is optimized through the refinement of the two-branch model established in the previous step ①, utilizing the following objective function:

$$L = \sum_{(x,y)} l(f(x, \mathbf{W_R}, \mathbf{W_T}), y) + \lambda \sum_{\gamma_R, \gamma_T} g(\gamma_R + \gamma_T) \quad (1)$$

where x and y denote the training input and label, while $\mathbf{W_R}$ and $\mathbf{W_T}$ represent the weights of the M_R and M_T branches, respectively. Also, f denotes the functionality of the two-branch model and l is the cross entropy loss for classification tasks. Note that the output of TBNet is derived from M_T, which predicts the inference result before returning it to the model users. During the training process, TBNet transfers the knowledge and updates the weights in two branches concurrently. Specifically, The training input x first runs the convolution computation with the first layer in TEE, and the resulting feature map is stored in its memory. Subsequently, the same training input x is processed through the first convolution layer in REE. The intermediate results from this computation are transferred to TEE and combined element-wise with the previously stored results (see ② in Fig. 1). Such computation is consistently applied across all subsequent layers. Then TBNet calculates the loss between the results from $\mathbf{W_R}$ and the target. Besides, γ_R and γ_T are the weights of the batch normalization (BN) layers, indicating the relative importance of each channel [9]. Here g is the sparsity-induced penalty on these weights, which is the L1 normalization, and λ balances these two terms. By minimizing L in Eq. 1, we can achieve knowledge transfer, as analyzed in Sec. 5.2. This optimization process not only distributes the knowledge of the victim model into two branches, but also encourages sparsity in BN weights, laying the groundwork for the subsequent pruning.

3.4 Step ③-⑤: Iterative Two-branch Pruning

While the current two-branch substitution model effectively prevents attackers from observing critical data transmissions (e.g.,

feature maps from TEE) and their direct utilization of the M_R, its deployment efficiency is impeded by the redundant parameters. Specifically, the secure branch (M_T) deployed in TEE should be optimized for both small model sizes and low computational demands to reduce secure memory utilization and execution latency. Although there are some methods designed for pruning the DNN model (e.g., [9]), they are designed for serial DNNs with only one branch and it is nontrivial to simultaneously prune two branches.

To mitigate these challenges, we propose an iterative two-branch pruning approach, incorporating a channel pruning strategy. This strategy enhances the computational efficiency and reduces memory requirements of M_T while modifying the architecture in M_R. We outline this pruning strategy in Alg. 1, which utilizes the sparsity-inducing regularization g in Eq. 1, to induce sparsity in the weights of the BN layers during the training phase. These sparse weights serve as the indicator, aiding in the identification of less important channels within the convolution layer. Therefore, the weights of BN layers for each channel in the M_R and M_T are extracted from the model at the first place in step ③.

Considering that the initial model architecture of M_R is identical to M_T and the intermediate feature maps result from M_R are added with the result of M_T, we combine the weights of the BN layers of M_R and M_T in channel scales (step ④ in Fig. 1) to create composite weights for the exact same architecture (line 4 in Alg. 1). The rationale for adding the weights together is that both M_R and M_T provide information and contribute to the inference, necessitating consideration of the channel's importance in two branches. This weight addition aligns with the addition of the intermediate feature map, allowing the composite BN weights to effectively represent the importance of the merged feature map. With the indication of composite weights, our framework prunes the less important channels accordingly. Following from line 5 to line 12 in Alg.1, our method determines a threshold for pruning based on the composite weights and the preset pruning ratio. Each channel's composite weights are then compared to this threshold. A pruning mask, consisting of 1's and 0's, is created to indicate whether a channel should be retained or pruned. This mask is applied simultaneously to both M_T and M_R, allowing for the pruning of the BN layer, convolution layer, and dense layer as required.

TBNet combines pruning and retraining in a single iteration, i.e., once a model undergoes pruning, it is also fine-tuned to recover the accuracy lost due to the reduction in model size. In this way, TBNet assesses the need for further pruning and refinement by comparing the accuracy decrease against a predefined budget for an acceptable accuracy drop. If the reduction in accuracy for M_T remains within the budget (i.e., θ_{drop} in Alg. 1), TBNet proceeds to the next round of pruning and retraining. However, if the accuracy drop exceeds θ_{drop}, the process halts and reverts to the prior state that satisfies the accuracy criteria.

3.5 Step ⑥: Rollback Finalization

Following step ③-⑤, the architectural configuration of M_T remains identical to that of M_R. However, from the security perspective, it is crucial to differentiate the architecture of the M_R from the M_T, since M_R is fully exposed to the attackers (see threat model in Sec. 2.2). Such an architectural difference is motivated by the need to safeguard the valuable architectural IP embodied in the TBNet, as it could potentially assist attackers in retraining to generate a high-performance model (see limitations of prior works in Sec. 2.3).

Algorithm 1 Iterative Two-branch Pruning

Require: M_T, M_R, accuracy drop budget θ_{drop}, pruning ratio p
1: **while** Accuracy drop < θ_{drop} **do**
2: $N \leftarrow$ Total number of channels in BN layers
3: $BN_T, BN_R \leftarrow$ BN Weights for each channel in M_T and M_R
4: $BN = BN_R + BN_T$ // Obtain composite weights
5: $T = sort(BN)[(N * p)]$
6: Initialize pruning mask $mask[N]$ with zeros
7: **for** $i = 1$ to N **do**
8: **if** $BN[i] > T$ **then**
9: $mask[i] \leftarrow 1$
10: **end if**
11: **end for**
12: Prune channels in M_T and M_R according to $mask$
13: **end while**
14: **return** M_T, M_R

To prevent the attacker from inferring the model architecture in TEE from that in REE, TBNet incorporates a rollback finalization step to establish a divergence, i.e., making $M_T \neq M_R$. Concretely, the architecture and weights in M_R rollback specifically to the state preceding the most recent pruning iteration. Given the ample resources in REE and the iterative nature of pruning (with a small pruning ratio), the incremental increase in the model size of M_R due to the rollback introduces minimal latency overhead. Meanwhile, the rollback of M_R to a previous state introduces additional parameters that contribute positively to overall performance. After receiving the feature map, M_T identifies and extracts the specific channel that aligns with their pre-stored feature map and runs the element-wise addition to combine the information together as the input for the next layer computation in TEE. This process ensures that the architecture in M_T remains confidential, effectively countering potential adversarial attempts to infer it.

4 EXPERIMENTAL EVALUATION

DNN Models. To validate the efficacy of TBNet, we conduct experiments with four DNN models, including VGG18 [15] and ResNet-20 [3] models that are trained on CIFAR10 and CIFAR100 [6] respectively. In particular, for ResNet20 with skip connections, M_R is initialized from the main branch (excluding skip connections), while M_T is initialized with the original architecture.

Hyperparameters. During the training of the model, we utilize the SGD optimizer with a learning rate of 0.1, a momentum of 0.9, and a weight decay of 1e-4. Also, we apply a learning rate scheduler, which reduces the learning rate to one-tenth every 100 epochs. The sparsity regularization λ in Eq. 1 is set to 10^{-4} in our experiment. For the iterative pruning process, we define the pruning ratio (p) as 10%, indicating a reduction of 10% of the total number of channels across the entire model.

Hardware Setup. We execute the implementation of TBNet on a Raspberry Pi 3 ModelB equipped with a Broadcom BCM2837 64-bit ARM Cortex-A53 Quad-Core Processor and 1GB of RAM. For

Datasets	DNN	Victim Acc. (%)	TBNet Acc. (%)	Attack Acc. (%)	Acc. Gap (%)
CIFAR10	VGG18	91.29	90.72	69.80	**20.92**
	ResNet20	92.27	91.68	10.00	**81.68**
CIFAR100	VGG18	67.41	68.37	42.64	**25.73**
	ResNet20	71.03	69.49	20.29	**48.54**

Table 1: The performance of TBNet and its protection against direct model usage (i.e., Attack Acc.).

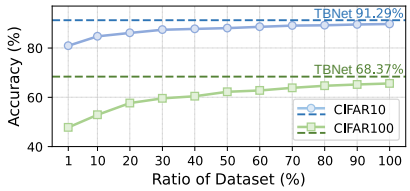

Figure 2: Accuracy of attackers fine-tuning the M_R of VGG18 under varying dataset availability.

generality, we employ Open Portable TEE (OP-TEE) [8], a versatile open-source TEE framework as the platform to verify TBNet.

4.1 Performance and Security Evaluation

We comprehensively evaluate TBNet from two perspectives: (1) **Performance**: whether TBNet can preserve the accuracy of the original model and (2) **Security**: whether an attacker can achieve comparable accuracy using M_R that is accessible in REE.

The experimental results in Tab.1 show that the inference accuracy TBNet is comparable to that of the victim model. Note this is a common "security-performance" trade-off, since we assume the original victim model from the vendor is highly optimized (see threat model in Sec. 2.2); thus, any further pruning will introduce performance loss. Leveraging the step ⑥ of TBNet, i.e., rollback finalization, the accuracy loss can be recovered once again. It is crucial to emphasize that the accuracy loss serves as the adjustable budget for achieving secure deployment, a parameter that defenders can tailor according to their specific requirements and trade-offs.

To evaluate the security protection of TBNet against model stealing, we compare two accuracy metrics: (1) the accuracy obtained by a benign user from the M_T output and (2) the accuracy achieved by the attacker when extracting M_R in REE and transplanting it for direct use. From Tab. 1, the achievable accuracy by the attacker is low, i.e., there is at least a 20% accuracy gap. Such a gap is more pronounced when the victim model is a ResNet model since the unavailability of data on residual blocks significantly impedes the attacker's ability to achieve high performance in direct use.

4.2 Protection Against Model Fine-tuning

To demonstrate the comprehensive and robust protection of TBNet, we assess its resilience against stronger attackers who attempt to fine-tune the model extracted from REE (M_R). Our evaluation simulates the attacker's fine-tuning process with varying training data availability, from 1% to 100%. As shown in Fig. 2, even if the attacker fine-tunes the M_R of VGG18 with 100% training dataset, s/he cannot achieve the performance as high as TBNet (e.g., 65.59% vs 68.37% on CIFAR100). This is because: (1) the post-prune architecture M_R is downgraded compared to the victim model and (2) the existence of M_T further enhances the overall performance of TBNet.

	TBNet	M_T	Acc. Drop
VGG18	91.29%	87.57%	**3.72%**
Resnet20	92.27%	89.41%	**2.86%**

Table 2: Accuracy comparison between the best possible M_T (i.e., re-training with all training data) and TBNet.

4.3 Hardware Efficiency

Here we demonstrate the hardware efficiency of TBNet compared to the case where victim models are fully executed in TEE (referred to as the **baseline**).

Memory Usage. Secure memory usage is a main limit while deploying DNN models within TEE, which makes optimizing secure memory usage essential. In our study, we assess the memory efficiency of TBNet by comparing the memory usage of M_T to the baseline approach. Our experiments are based on the premise that the model vendor already has a compact model, which cannot be further pruned without loss of accuracy. This condition underscores the efficacy of TBNet, which achieves up to a 2.45x reduction in memory usage. This improvement, demonstrated using VGG18 and CIFAR10, can be attributed to the elimination of redundancies inherent in the two-branch model architecture, even if the victim model is well-pruned. These experimental results demonstrate TBNet's potential in generating secure and lightweight DNN on the edge.

Inference Latency. We also evaluate the performance of TBNet by measuring the inference latency of its hardware implementations. Specifically, the baseline implements the entire victim model in TEE, while TBNet only implements the secure branch (M_T) in TEE. We show the experimental results in Tab.3, which indicates that the implementation of TBNet helps to reduce the inference latency up to 1.22x compared with the baseline method. This demonstrates that our protection method is more advantageous compared with the baseline method, making it a better solution for time-sensitive applications. Note that the result of latency reduction is the minimized results achievable by TBNet, which does not apply any other optimization strategies, we discuss the optimization ways to accelerate the model inference in Sec. 5.3.

5 DISCUSSION

5.1 Necessity of Unsecured Branch

While TBNet demonstrates superior performance over the existing solutions, by using the two-branch DNN model architecture, the role of M_R in REE remains unexplored. In other words, it is unclear whether M_T alone could match the performance of TBNet without the support and information from M_R. To investigate the significance of M_R, we remove its architecture and weights from TBNet and leave the M_T as the only branch of the TBNet. Then retrain M_T with the entire training dataset to evaluate its optimal performance. Tab. 2 presents a comparison of accuracy between TBNet and its single branch M_T, across two different model architectures, VGG18 and Resnet20, on the CIFAR10 dataset. Results indicate a decline in

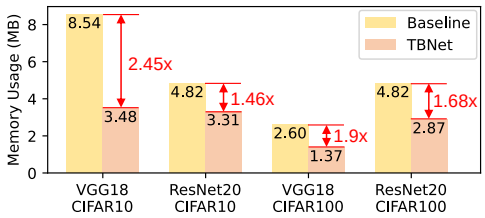

Figure 3: The comparison of memory usage in TEE.

	Baseline	TBNet	Reduction
VGG18	2.3983	1.9589	**1.22×**
ResNet20	3.7425	3.2667	**1.15×**

Table 3: The inference latency (s) of models for CIFAR10.

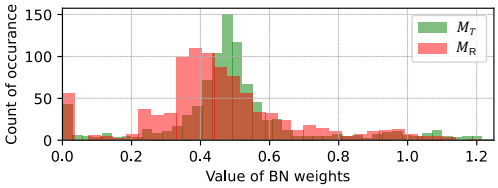

Figure 4: Distribution of BN weights in M_T and M_R after knowledge transfer.

performance accuracy (up to 3.72%) when operating without the unsecured branch (M_R), signifying that the intermediate results transmitted from REE are necessary for the overall performance.

5.2 Analysis of Knowledge Transfer

As discussed in Sec. 3.3, one important contribution of TBNet is to reduce the model confidentiality exposure in REE, i.e., by transferring partial knowledge from M_R to M_T. In Fig. 4, we visualize the distributions of BN weights in M_R and M_T, respectively, after applying the knowledge transfer. These experimental results demonstrate that the knowledge from the victim model is effectively distributed between M_R and M_T. Recalling that BN weights are used to indicate channel importance, it is noteworthy that, on average, channels in M_R demonstrate lower values compared to those in M_T, considering that channels with smaller BN weights contribute less to the model performance [9].

5.3 Potential Extension of TBNet

The experimental results in Tab. 3 demonstrate the minimal achievable inference latency reduction by TBNet. In real-world scenarios, the performance of TBNet can be further enhanced by using diverse techniques, such as multiple thread execution, GPU acceleration, and specific libraries for DNN, all of which can greatly reduce the execution time of M_R in REE. In other words, our proposed TBNet framework can be compatible with and enhanced by any existing hardware acceleration techniques. The rapid development of AI hardware accelerators highlights the great potential of our proposed two-branch methodology, which enables secure collaborative inference between REE and TEE for both performance and security.

6 CONCLUSION

This paper presents TBNet, a novel neural architectural defense framework to facilitate efficient DNN model protection using TEE. TBNet generates a two-branch substitution model and deploys the secure branch within TEE for security enhancement. It also minimizes the performance overhead by offloading the unsecured branch to REE. Experimental results from diverse model architectures and datasets demonstrate that, TBNet reduces the execution time of the model inference by 1.22×, while protecting the functionality and architecture of the model against stealing even if the attacker has entire training datasets.

ACKNOWLEDGEMENT

This work is supported in part by the U.S. National Science Foundation under Grants OAC-2319962, CNS-2239672, CNS-2153690, CNS-2326597, and CNS-2247892.

REFERENCES

[1] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX Security'19*.
[2] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepAttest: An end-to-end attestation framework for deep neural networks. In *ISCA'19*.
[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. [n. d.]. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016*.
[4] Mihailo Isakov, Vijay Gadepally, Karen M Gettings, and Michel A Kinsy. 2019. Survey of attacks and defenses on edge-deployed neural networks. In *HPEC'19*.
[5] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *USENIX Security'20*.
[6] Alex Krizhevsky and Geoff Hinton. 2010. Convolutional deep belief networks on cifar-10. *Unpublished manuscript* 40, 7 (2010), 1–9.
[7] Arm Limited. 2023. TrustZone for Cortex-M. https://www.arm.com/technologies/trustzone-for-cortex-m
[8] Linaro Limited. 2019. Open Portable Trusted Execution Environment. https://www.op-tee.org
[9] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *ICCV'17*.
[10] Ziyu Liu, Yukui Luo, Shijin Duan, Tong Zhou, and Xiaolin Xu. 2023. MirrorNet: A TEE-Friendly Framework for Secure On-Device DNN Inference. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.
[11] Xiaoxuan Lou, Shangwei Guo, Jiwei Li, and Tianwei Zhang. 2022. Ownership verification of dnn architectures via hardware cache side channels. *TCSVT* (2022).
[12] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In *MobiSys'20*.
[13] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. 2022. SoK: machine learning with confidential computing. *arXiv preprint arXiv:2208.10134* (2022).
[14] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2021. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–37.
[15] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[16] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. [n. d.]. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *SP'23*.
[17] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. 2021. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *USENIX Security'21*.
[18] Peter M VanNostrand, Ioannis Kyriazis, Michelle Cheng, Tian Guo, and Robert J Walls. 2019. Confidential deep learning: Executing proprietary models on untrusted devices. *arXiv preprint arXiv:1908.10730* (2019).
[19] Xueli Xiao, Thosini Bamunu Mudiyanselage, Chunyan Ji, Jie Hu, and Yi Pan. 2019. Fast deep learning training through intelligently freezing layers. In *iThings'19*.
[20] Ziqi Zhang, Lucien KL Ng, Bingyan Liu, Yifeng Cai, Ding Li, Yao Guo, and Xiangqun Chen. 2022. Teeslice: slicing dnn models for secure and efficient deployment. In *AISTA'22*.