Secure and Efficient Mobile DNN Using Trusted Execution Environments

Bin Hu
bh439@scarletmail.rutgers.edu
Rutgers University

Yan Wang
y.wang@temple.edu
Temple University

Jerry Cheng
jcheng18@nyit.edu
New York Institute of Technology

Tianming Zhao
tzhao1@udayton.edu
University of Dayton

Yucheng Xie
yx11@iupui.edu
Indiana University-Purdue University Indianapolis

Xiaonan Guo
xguo8@gmu.edu
George Mason University

Yingying Chen
yingche@scarletmail.rutgers.edu
Rutgers University

ABSTRACT

Many mobile applications have resorted to deep neural networks (DNNs) because of their strong inference capabilities. Since both input data and DNN architectures could be sensitive, there is an increasing demand for secure DNN execution on mobile devices. Towards this end, hardware-based trusted execution environments on mobile devices (mobile TEEs), such as ARM TrustZone, have recently been exploited to execute CNN securely. However, running entire DNNs on mobile TEEs is challenging as TEEs have stringent resource and performance constraints. In this work, we develop a novel mobile TEE-based security framework that can efficiently execute the entire DNN in a resource-constrained mobile TEE with minimal inference time overhead. Specifically, we propose a progressive pruning to gradually identify and remove the redundant neurons from a DNN while maintaining a high inference accuracy. Next, we develop a memory optimization method to deallocate the memory storage of the pruned neurons utilizing the low-level programming technique. Finally, we devise a novel adaptive partitioning method that divides the pruned model into multiple partitions according to the available memory in the mobile TEE and loads the partitions into the mobile TEE separately with a minimal loading time overhead. Our experiments with various DNNs and open-source datasets demonstrate that we can achieve 2-30 times less inference time with comparable accuracy compared to existing approaches securing entire DNNs with mobile TEE.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

ASIA CCS '23, July 10–14, 2023, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0098-9/23/07…$15.00
https://doi.org/10.1145/3579856.3582820

KEYWORDS

Network Pruning, TEE, DNN, Security in Machine Learning

ACM Reference Format:
Bin Hu, Yan Wang, Jerry Cheng, Tianming Zhao, Yucheng Xie, Xiaonan Guo, and Yingying Chen. 2023. Secure and Efficient Mobile DNN Using Trusted Execution Environments. In *ACM ASIA Conference on Computer and Communications Security (ASIA CCS '23), July 10–14, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3579856.3582820

1 INTRODUCTION

Deep Neural Networks (DNNs) have achieved remarkable successes in a broad range of mobile applications such as image recognition [24], mobile entertainment [18], and VR/AR applications [16], due to their performance scalability and self-adaptiveness [28]. However, recent studies show that DNNs and their memorized information on mobile devices are subject to a variety of security threats, such as Membership Inference Attack (MIA) [50], Data Reconstruction Attack (DRA) [7], and Property Inference Attack (PIA) [33]. Thus, protecting DNNs and preserving their data privacy are critical for developing mobile applications on resource-limited mobile devices.

In recent years, mobile devices have been increasingly equipped with small hardware-isolated Trusted Execution Environments (TEEs) responsible for security-critical operations. A TEE employs the hardware security primitive to protect sensitive contents (e.g., private data and DNN architecture) executed in it even if an attacker has compromised the operating system. However, it is very challenging to execute DNNs using TEEs on mobile devices (mobile TEEs) due to the huge gap between the high demand for resources (i.e., memory and computing power) of DNNs and the stringent resource constraints of mobile TEEs. For instance, ResNet-50 [52] on the ImageNet [47] dataset demands over 150MB of memory, whereas the typical implementation of the mobile TEE only has up to 16MB of memory (i.e., Arm TrustZone Cortex-A [49]). Although it is physically feasible to deploy more resources into the secure world (e.g., memory), doing so is directly against the security principle of maintaining a small Trusted Computing Base (TCB) and would result in an increased level of exploitable security vulnerabilities [2, 6, 55]. For example, Cerdeira *et al.* [2] discovered that

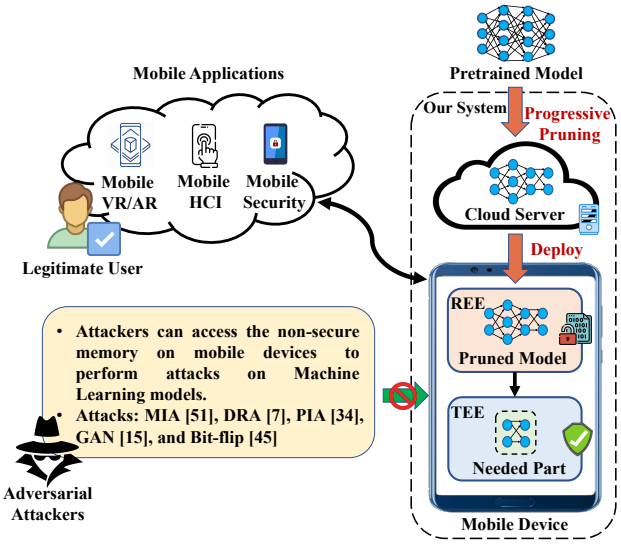


Figure 1: Our approach takes two steps to enable secure and efficient DNN execution in mobile TEE to defend against Machine Learning attacks on mobile devices.

using large sizes of TCB (e.g., Qualcomm TEE) will expose more potential vulnerabilities to attackers. Thus, the industry followed this principle and limited the resources (e.g., secure memory) in mobile TEE (e.g., 8MB-16MB for Raspberry Pi 3 Model B, HiKey family, and Juno) [30, 36] to keep the TCB as small as possible. For the cloud TEE (i.e., Inter SGX [3]), the physically protected memory is also limited to 128MB set in BIOS. Furthermore, the TEE-supported operating system also limits the memory size (e.g., OP-TEE [32] for 32MB). In this work, we aim to develop a secure and efficient DNN execution framework via mobile TEE, which adopts this security principle with a small mobile TEE size (i.e., 16 MB) to decrease the level of exploitable security vulnerabilities.

Several existing works [36, 55] have tried to address the above challenge of securing DNN execution on mobile devices by executing partial DNN (e.g., the first few layers) in a mobile TEE and the rest of the DNN outside the mobile TEE. However, these approaches still have traceable information outside TEE that can be compromised by attackers (e.g., Bit-Flip Attack [44]). Recently, researchers propose to secure the entire DNN execution by entirely or partially offloading the DNN executions to the TEE on cloud servers (i.e., cloud TEEs), which have more resources than mobile TEEs. For example, HybridTEE [6] puts the first few layers of DNNs in a mobile TEE and the rest in a cloud TEE to reduce the workload of the mobile TEE. Although these approaches can meet the desired security requirement of protecting the entire DNN execution, they heavily rely on network connections which may be unpredictable and unstable. Furthermore, these existing approaches suffer from communication degradation since a large number of intermediate results need to be transmitted from mobile devices to the cloud server during the DNN execution. Although the advancement of mobile network technologies (e.g., 5G) can mitigate communication degradation, they still involve concerns about data privacy and network connection availability. More importantly, none of the existing works consider the inference efficiency when leveraging mobile TEEs, which is a critical factor impeding the deployment of DNNs in resource-constrained mobile TEEs.

In this work, we develop an innovative, secure, and efficient DNN execution framework using mobile TEEs. Our approach is the first framework that allows resource-constrained mobile devices to securely execute the entire DNN with high accuracy and low latency. Unlike existing approaches [36, 41] that only optimize DNN inference accuracy in mobile TEEs, we aim to achieve high inference accuracy while minimizing the inference time and execution overhead. Toward this end, we develop the progressive pruning and memory optimization methods to gradually identify and remove redundant neurons (i.e., filters in convolutional layers and nodes in fully connected layers) of a target DNN. Pruning is more suitable and necessary to run a large-size DNN in memory-constrained mobile TEE compared with other compression techniques (e.g., quantization [58], low-rank factorization [34], and knowledge distillation [56]) since it has the advantages of low power consumption, memory efficiency, and provides faster inference with minimal accuracy compromise. A novel adaptive partitioning and loading method is designed to separate the pruned model into optimal-size partitions according to the available memory of the mobile TEE and execute them in the mobile TEE efficiently. Figure 1 illustrates the basic idea of our approach. We assume machine learning attackers can access non-secure memory on mobile devices. The attackers aim to compromise the integrity and confidentiality of the DNN inference on mobile devices. With our unique design of the fine-grained pruning and resource-aware partitioning, our approach enables various time- and privacy-sensitive mobile applications that existing approaches cannot support, including mobile VR/AR applications [16] involving users' private location information, mobile human-computer interaction (HCI) applications [4] involving users' daily activities, voice signatures, and hand gestures, and mobile security applications [14] using security tokens and biometrics.

There are many challenges in realizing such an optimization mechanism for secure DNN inference using mobile TEE. To name a few, designing an effective and efficient pruning method to enable DNN execution in TEEs is nontrivial. It requires the pruning method to accurately identify the redundant model parameters with minimal retraining iterations. In addition, existing pruning approaches (e.g., [60, 61]) cannot deallocate the memory space of pruned parameters because they adopt the static memory allocation technique (e.g., static array in TensorFlow and PyTorch). Thus, the pruned DNNs require the same memory and computational cost as pre-trained models, which cannot fit into a resource-constrained mobile TEE. Moreover, no existing work supports executing the entire DNN of various sizes (e.g., small size and large size) in the mobile TEE because they cannot efficiently partition and load multiple layers into mobile TEEs. Furthermore, most DNNs use a large size of memory (e.g., ResNet-50 requires 150MB) or have many layers (e.g., GoogLeNet has 57 layers). Thus, it is hard to execute DNNs in mobile TEE efficiently if we use existing layer-by-layer partitioning and loading approaches, which cannot support various time- and privacy-sensitive mobile applications.

To address these challenges, we jointly employ a structure pruning approach to reduce memory and computational cost. To maintain the high inference accuracy, we propose *salient score* to measure

the global importance of each neuron. In contrast to simply removing a neuron in each pruning iteration, we design a *scale gate* to gradually adjust the impact of neurons on the network, which helps to reduce the re-train iterations for achieving a reasonably high model accuracy. We design a low-level re-coding mechanism that facilitates the irregular neuron patterns to a regular one and deallocates the memory storage for pruned neurons. Thus, the optimized DNNs can use reduced memory and computation costs and achieve inference speedup on resource-constrained mobile devices. In addition, our approach adopts an adaptive partitioning method that divides the DNNs into multiple parts, which contain as many layers as possible while satisfying the memory constraint of the mobile TEE. The proposed multi-layer loading method can load needed parts into mobile TEEs with minimal loading iterations. Compared to the existing approaches of only loading and executing one layer of DNN in the TEE in each iteration, our approach significantly reduces loading iterations and inference time overhead, especially for large DNNs.

We summarize our main contributions as follows:

- We propose a novel structured pruning-based optimization mechanism to enable the secure execution of DNNs in resource-constrained mobile TEEs. Our optimization mechanism is the first of its kind that can accurately identify redundant neurons, significantly reduce the actual memory usage of DNNs, and efficiently execute all DNN layers in mobile TEEs.
- We develop a fine-grained structured pruning method that gradually analyzes and adjusts the impact of the neurons on the DNN's loss. As a result, the pruning method can accurately identify redundant neurons having minimal impact on the inference accuracy with significantly reduced retraining iterations.
- We develop a low-level re-coding optimization mechanism that transfers irregular neuron patterns to regular ones and deallocates the memory storage of pruned neurons. Such an approach reduces the memory usage of DNNs and facilitates DNN execution in resource-constrained mobile TEEs.
- We design an adaptive partitioning method that dynamically partitions the pruned DNNs into the partition containing as many layers as possible while fitting the memory constraint of the mobile TEE. The proposed adaptive partition method significantly reduces the loading iterations and inference time overhead compared to the existing layer-by-layer loading approaches.
- We put great effort into implementing the first framework using Darknet [45], DarkneTZ [36], and OP-TEE. The framework can significantly reduce the effort of developing future research on DNN optimization in Arm TrustZone. We report the implementation details and our experience in this work. The source code of the framework is shared through the public repository.
- Extensive experiments are conducted to evaluate the effectiveness of our approach with various DNNs and datasets. Evaluation results demonstrate that we can achieve 2-30 times less inference time with comparable accuracy compared to existing approaches (i.e., LL-TEE, TrustedDNN) securing entire DNNs in mobile TEE.

2 BACKGROUND

2.1 Trusted Execution Environment (TEE)

The recent developments of TEEs have enabled the opportunity to secure the computation-intensive workload. A TEE enables the creation of a secure area on the main processor that provides strong confidentiality and integrity guarantees to any data and codes it stores or processes. Over the last few years, significant research and industry efforts have been devoted to developing secure and programmable TEEs for high-end devices and mobile devices (e.g., Arm TrustZone and AMD SEV [39]). The most popular TEE that provides access protection to mobile devices is Arm TrustZone.

ARM TrustZone divides the entire System-on-Chip resources into two "worlds": The normal world executing vulnerable and untrusted applications is referred to as the Rich Execution Environment (REE). The secure world executing secured applications is referred to as TEE. Accordingly, a security-sensitive application divides itself into two components: an REE-side component called Client Application (CA) and a TEE-side component called Trusted Application (TA). All sensitive operations are isolated in TA, which usually runs on a Trusted OS inside the TEE. By leveraging the TrustZone's hardware-assisted isolation, the confidentiality and integrity of TAs are protected from the untrusted REE. To overcome the security and privacy issues of DNN inference, we propose to execute the entire DNNs inside the TrustZone by leveraging hardware-based protection mechanisms.

2.2 Neural Network Pruning

Neural network pruning aims to compress the DNNs by removing the redundant parameters to facilitate the dense network into a sparse one. Pruning approaches can be categorized as non-structured pruning and structured pruning. Non-structured pruning [27, 59] prunes weights achieving high pruning rates and promising accuracy, but the resulting pruned model has sparse weight matrices. As a result, we can not skip the computational process for pruned weights to reduce the overall computational cost in real implementations [40]. Structured pruning [57, 61] removes the whole filter with a regular structure form, making it more suitable to obtain direct acceleration on mobile devices.

Since current mobile TEEs have limited memory (e.g., 8-16MB), it is hard to load and execute the entire DNN in the mobile TEE simultaneously. For example, the Arm Trustzone used for this study offered 16MB of memory, whereas VGG-16 [51] on the ImageNet [47] dataset demand over 630MB of memory. To overcome this issue, we develop a structured pruning method that removes the unimportant neurons of DNNs to compress their size. As a result, the memory and computation demand of DNNs can be significantly reduced, and the pruned DNNs can be deployed into resource-constrained mobile TEEs while maintaining high accuracy.

3 METHODOLOGY

3.1 Threat Model and Assumptions

We consider an adversary that can access the non-secure memory on mobile devices (e.g., an actual user with authenticating, malicious third-party software, or compromised OS). The goal is to compromise the confidentiality and integrity of the target DNN model during the inference, including: 1) modifying the input, weight, and intermediate results to alter the inference results; and 2) reconstructing the DNN model by using information (e.g., input, intermediate results, weight) obtained during inference. We only trust mobile TEE to guarantee the integrity and confidentiality of the data, DNN model, and software. We do not consider the side-channel attack

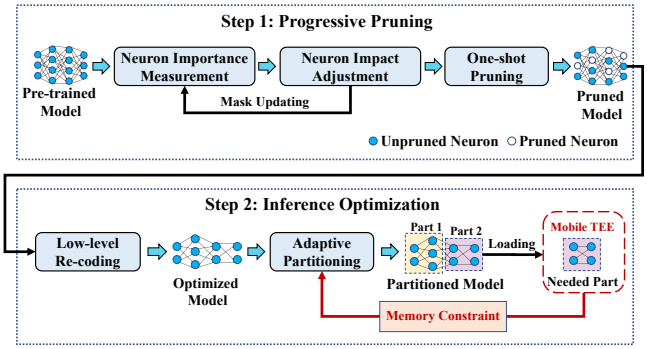


Figure 2: Illustration of our proposed approach.

leveraging the memory information inside the mobile TEE to compromise the DNNs.

We assume that training and pruning processes can guarantee the privacy and integrity of DNNs on the server. We also assume the input data and pre-trained DNN model deployed on REE are encrypted using secure protocols (e.g., MD5 hash [46]). TA will decrypt the loaded part and then execute the DNN inference inside mobile TEE. Decryption/encryption keys are securely provisioned into the secure/insecure world. Several existing works [29, 36] have demonstrated that such decryption and encryption processes are practical in use, and the decryption only introduces a few time overheads (i.e., less than 5% of inference time) in real-world implementations.

3.2 Design Goals

Our design aims to achieve the following goals:

Securing the entire DNN inference. We aim to protect inference process and results. Moreover, we seek to safeguard DNN weights and intermediate results generated during inference since malicious attackers can use it to reconstruct input data, infer sensitive information, and generate unexpected outputs.

Reducing the DNN inference time overhead. Inferencing DNNs on mobile TEE introduces significant inference time overhead due to its memory and computation constraints. The proposed approach aims to improve memory and inference efficiency to generate minimal inference time overhead while protecting the entire DNN inference using the mobile TEE.

Maintaining the high inference accuracy. Our goal is not to achieve a higher pruning rate compared to the existing state-of-the-art pruning approaches (e.g., [28, 38, 54]). We aim to maintain the high inference accuracy for the pruned DNNs while achieving as much as more pruning rate. Thus, large DNNs could be compressed and deployed on mobile TEE with high inference accuracy.

3.3 Design Overview

We aim to secure the entire DNN inference on mobile TEE and improve the inference efficiency. Considering the limited memory and computational resources of mobile TEE, we design a two-step mechanism that first prunes the redundant neurons to reduce the memory and computational cost of the DNN and then optimizes the pruned DNN to accelerate the inference speed on the mobile TEE. The flow of the proposed framework is illustrated in Figure 2.

First, we perform a *Progressive Pruning* to suppress the impact of redundant neurons on the network gradually and then remove them with the minimal accuracy loss. To identify the redundant neurons globally, we propose the neuron *salient score* in Section 3.4 corresponding to the impacts on the DNN's performance. To enable neurons' gradual adjustment in importance, we design a *scale gate* and update masks associated with each neuron using three different strategies (amplify, hold, and suppress) according to their different levels of salient score. We identify the unimportant neurons by examining the value of masks with a threshold approach. The pruning iteration stops when the number of unimportant neurons reaches a target compression ratio. Finally, we perform a one-time pruning to remove the identified unimportant neurons. At the target compression ratio, our approach performs the retrain to recover the DNN's accuracy. Since the unimportant neurons are removed with minimal impact on the DNN, the accuracy drop in the DNN caused by the removed neurons is smaller than in the traditional direct pruning approach.

Next, we propose an *Inference Optimization* mechanism to optimize the DNN inference for reducing the memory storage and accelerating inference. Specifically, we develop a low-level re-coding optimization method that reorders the neurons to a regular pattern and removes the occupied storage of pruned neurons by using the low-level programming technique (i.e., C programming). Compared with the existing pruning approaches without using such an optimization method, our approach achieves memory reduction in a real-world implementation since the allocated memory of pruned neurons is de-allocated. To minimize the mobile TEE loading iterations and time, we design an adaptive partitioning mechanism (i.e., layer-wised and sub-layer) that partitions the optimized DNN into multiple parts. The layer-wised approach ensures each part contains as many layers as possible and fits into the available size of the mobile TEE. The sub-layer approach divides the one layer into multiple parts with the maximum size if the size of one layer exceeds the available size of the mobile TEE. Thus, the entire optimized DNN could be loaded and run on mobile TEE separately with minimal loading iterations and time. During the DNN inference, when the current part completes its execution, our approach keeps the necessary results as the input for the next needed part to load. The process is repeated until the final output is generated. The details of each step are described in the following sections.

3.4 Progressive Pruning

In this paper, we employ structure pruning to jointly reduce the memory demand and improve the inference efficiency. Consider a set of training examples $\mathcal{D} = \{X = \{x_0, x_1, \cdots, x_z\}, Y = \{y_0, y_1, \cdots, y_z\}\}$, where x and y represent inputs and target outputs, z is the size of training set, respectively. The neurons $\mathcal{N} = \{n_0, n_1, \cdots, n_k\}$ are optimized to minimize the cross-entropy loss $\mathcal{L}(\mathcal{N}; \mathcal{D})$ between the prediction results and ground truth. During pruning, we refine a subset of neurons $\mathcal{N}'$ that tries to preserve the accuracy of the original network as much as possible. This corresponds to an optimization equation:

$$\min_{\{\mathcal{N}'\}} |\mathcal{L}(\mathcal{N}'; \mathcal{D}) - \mathcal{L}(\mathcal{N}; \mathcal{D})| \quad s.t. \ \| \mathcal{N}' \|_0 \leqslant \theta, \tag{1}$$

where $\mathcal{N}'$ is a subnet of $\mathcal{N}$, θ is a target pruning rate, $||\cdot||_0$ is the L_0 norm. To minimize the difference in accuracy between the full and pruned DNN, we need to identify the importance of each neuron correctly. Thus, we design a mask M, which is a vector of auxiliary indicator variables $m_i \in \{0, 1\}$ for every neuron in $\mathcal{N}$. With this,

we leverage M to measure the importance of $\mathcal{N}$ to the DNN and decide whether to prune them or not based on their corresponding mask values in M. Equation 1 can be modeled as:

$$\min_{\{M\}} |\mathcal{L}(\mathcal{N} \odot \widehat{M}; \mathcal{D}) - \mathcal{L}(\mathcal{N} \odot M; \mathcal{D})| \quad s.t. \ \| M \|_0 \leqslant \theta, \quad (2)$$

where $\widehat{M}$ are the masks for the pruned model, in which the value of the corresponding pruned neuron is 0, $\odot$ is the Hadamard multiplication. To finalize the $\widehat{M}$ of the pruned model, we propose to capture the impact of loss with and without i^{th} neuron as:

$$\Delta\mathcal{L}_i(\mathcal{N}; \mathcal{D}) = \mathcal{L}(\mathcal{N} \odot \widehat{M}_i; \mathcal{D}) - \mathcal{L}(\mathcal{N} \odot \widetilde{M}_i; \mathcal{D}), \quad (3)$$

where $\widehat{M}_i$ and $\widetilde{M}_i$ are the masks with and without the i^{th} neuron, respectively. However, Equation 3 is computationally expensive as it requires millions of forward passes over the dataset. To solve this problem with reasonable computing cost, we use the partial derivation of the loss function with respect to the i^{th} mask M to approximate $\Delta\mathcal{L}_i(\mathcal{N}; \mathcal{D}) \approx \sigma_i(\mathcal{N}; \mathcal{D}) = \partial\mathcal{L}(\mathcal{N} \odot M; \mathcal{D})/\partial M_i$.

Based on $\sigma_i(\mathcal{N}; \mathcal{D})$, we define a neuron *salient score* ε_i to measure the importance of neurons as [19]:

$$\varepsilon_i = |\sigma_i(\mathcal{N}; \mathcal{D})| / \sum_{j=1}^{k} \sigma_j(\mathcal{N}; \mathcal{D}). \quad (4)$$

The neuron's salient score ε_i would reflect the importance of the i^{th} neuron to the DNN since the magnitude of ε_i indicates the impact of i^{th} neuron on the network loss. Existing pruning methods directly remove the redundant neurons by setting the binary masks with the value of 0 in a one-shot pruning manner. Such approaches have coarse-grained resolutions on discovering neurons' importance and may result in degraded performance with possible loss of essential features by mistake. Moreover, the degraded performance requires more iterations in the retraining step to recover the accuracy. Since the estimation of neural salient score is the key to the accuracy of the pruning approach, we propose to iteratively identify the salient score and adjust its value.

The proposed progressive pruning consists of the following three steps: 1) iterations of identifying neuron salient score and adjusting their impact on the network; 2) stop pruning after reaching the target accuracy drop; 3) fine-tune the network until convergence on the target task. In contrast to the existing approaches, we analyze and adjust the neurons' salient score in a fine-grained way by gradually updating the mask. As a result, our approach can ensure minimal accuracy drop while removing the neurons, facilitating iteration reduction in the retraining process.

Specifically, in each pruning iteration, we derive a list of salient scores $\varepsilon = \{\varepsilon_1, \cdots, \varepsilon_k\}$ for each neuron using Equation 4 and sort the list by the descending order: $\varepsilon^s = \{\varepsilon^s_1, \cdots, \varepsilon^s_k\}$. The ε^s is then split into three sublists based on their impact on the network: $\varepsilon^s_{amp} = \{\varepsilon^s_1, \cdots, \varepsilon^s a\}$, $\varepsilon^s_{hold} = \{\varepsilon^s_{a+1}, \cdots, \varepsilon^s b\}$, and $\varepsilon^s_{sup} = \{\varepsilon^s_{b+1}, \cdots, \varepsilon^s_k\}$, where a and b are boundary index of three sublists. To dynamically track the impact of each neuron on the network with various input samples and adjust their impact in a fine-grained way, we design a dynamic updating strategy (i.e., amplify, hold, and suppress) for updating corresponding mask by using a *scale gate* α_i: $m'_i = \alpha_i \times m_i$. We determine the value of α_i by using the following strategy:

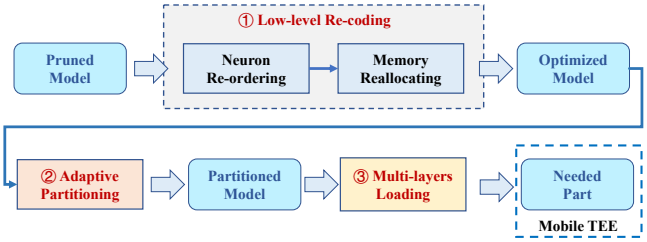


Figure 3: Our approach takes three steps to optimize the pruned model to reduce the memory size of the DNN model and ensure the minimized inference time on mobile TEE.

$$\alpha_i = \begin{cases} \alpha_{i,amp}, & \alpha_{i,amp} > 1, \quad when\ \varepsilon^s_i\ in\ \varepsilon^s_{amp} \\ \alpha_{i,hold}, & \alpha_{i,hold} = 1, \quad when\ \varepsilon^s_i\ in\ \varepsilon^s_{hold} \\ \alpha_{i,sup}, & 0 < \alpha_{i,sup} < 1, \quad when\ \varepsilon^s_i\ in\ \varepsilon^s_{sup}. \end{cases} \quad (5)$$

Our approach ensures the impact of unimportant neurons on the network decays gradually and will not result in significant degradation of the accuracy during the pruning process. After mask updating, we compute the ratio of unimportant neurons based on the masks that are less than a cut-off threshold β. If the ratio exceeds a target compression ratio θ, we stop updating the masks and performs a one-shot pruning on the unimportant neurons corresponding to the masks that are less than the cut-off threshold. Eventually, we perform the retraining process to fine-tune the model accuracy by recovering certain weights of neurons. In this work, we empirically determine the scale gate, dividing indices, cut-off thresholds, and target compression ratios based on input models.

3.5 Inference Optimization

Although the redundant neurons are removed by progressive pruning, the pruned neuron still occupies the memory space in a real-world implementation. That is because existing pruning approaches adopt static memory allocation techniques (e.g., static array in Tensorflow or PyTorch) to store the neurons. Such static memory allocation techniques cannot really delete the elements from the memory space. Furthermore, the size of pruned DNN may still suffer from the memory constraint of mobile TEE. To solve these memory problems, we employ a three-stage inference optimization mechanism to reduce the memory occupation: 1) *Low-level re-coding*: we shape the irregular neuron pattern to the regular pattern and deallocate the memory space of the pruned neurons; 2) *Adaptive partitioning*: the DNN is partitioned into multiple parts. Each part contains as many layers as possible while fitting the constraint of memory size of mobile TEE; 3) *Multi-layers loading*: the trust application (TA) loads the needed part, which contains as many layers as possible to infer the result on mobile TEE with minimal loading time overhead. Figure 3 illustrates the flow of the proposed inference optimization mechanism. We describe the details in the rest of this section.

Low-level Re-coding: In traditional pruning approaches, the pruned neurons still occupy the memory storage because they utilize the static data structure to store the neurons. So the memory size and computational cost cannot be reduced in a real-world implementation. To address this issue, we propose a low-level recoding

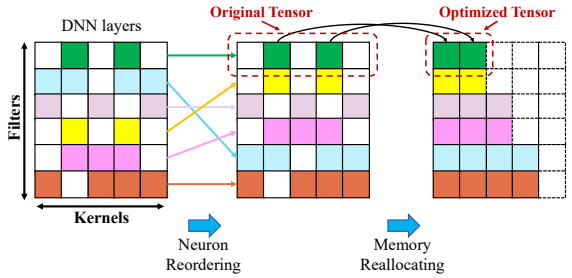


Figure 4: An example of neuron reordering and memory reallocating optimization.

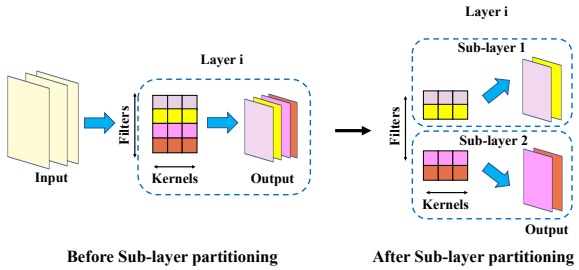


Figure 5: Illustration of sub-layer partitioning approach for convolutional layer.

optimization to eliminate pruned neurons' memory and computational cost. Specifically, we first extract the sparsity information from pruned DNNs. It includes neuron patterns and connectivity-related information (e.g., the pruned neuron patterns presented in each layer, the connection between the neuron and input/output channels, the input and output sizes, etc.).

Next, we leverage the extracted information to perform the neuron reordering, which facilitates irregular neuron patterns to the regular ones [40]. In particular, we first organize the filters with a similar number of kernels together to improve inter-thread parallelization and order the same kernels in a filter together to improve intra-thread parallelization. Figure 4 explains neuron reordering with a simplified example. Here, a matrix represents a convolutional layer of the DNN model, where each cell is a kernel. Empty kernels are the ones pruned by our pruning approach. The kernels in the same row belong to the same filter and are marked with the same color. Before the reordering, kernels with different numbers are randomly distributed in one layer. After the reordering, the filters with the same number of kernels are grouped together.

Then, we utilize the low-level programming (i.e., C/C++) to optimize the data structure (e.g., array) of the pruned DNNs, storing the kernel/filter to a new optimized one. Specifically, we manually redefine the new data structure to replace the original one. The new data structure allocates the contiguous memory space with regular memory patterns for the remaining neurons and releases the memory storage for pruned ones. Our approach doesn't need any dedicated designed hardware to accelerate the inference speed or reduce the memory size of pruned models in a real-world implementation.

Adaptive Partitioning: After low-level re-coding optimization, the pruned DNN may still exceed the memory constraint of the mobile TEE. Furthermore, the current programs that utilize the deep learning framework (e.g., PyTorch) allocate the memory for all parameters at the beginning of the execution at once [21]. Such implementation results in a high paging rate and time overhead when DNN size exceeds the available memory size [41]. Thus, we propose a novel adaptive partitioning mechanism to divide the optimized DNN into multiple parts. Each part is selected to be sufficiently small to fit in the limited mobile TEE memory. By executing each part separately, the entire DNN could be computed in pieces on mobile TEE. More importantly, each part should contain as many layers as possible to achieve the minimal inference time overhead introduced by multiple loading iterations. To achieve these goals, we develop layer-wised partitioning and sub-layer partitioning methods to dynamically partition the optimized DNNs based on the memory requirements of DNN inference and mobile TEE. Although an existing work, Vessels [21] developed an on-demand parameter loading method to load the needed parameters to the cloud TEE during DNN inference, such an approach aiming to minimize the memory footprint of the DNN model wastes many available memory resources of TEE. As a result, it introduces the high loading iterations and inference time overhead. In contrast, our approach can fully utilize the limited memory resources of mobile TEE and achieve fewer loading iterations and inference time overhead.

Layer-wised Partitioning: The optimized DNN is broken into multiple parts containing one or more consecutive layers. Each part should have as many layers as possible while fitting the memory constraint of mobile TEE to have the minimal loading time iterations. Since the mobile TEE is designed to conceal sensitive information (e.g., memory usage), we estimate the available size of mobile TEE and derive the maximum memory size of each part. We assume that the mobile TEE is only used for DNN inference and the size of secure memory Mem_{TEE} is fixed, which includes Mem_{TA} for the TA and Mem_{RUN} for TEE run-time. We first extract the sparsity information (e.g., neuron pattern, the memory size of each layer, the memory size of input/output, and the memory size of intermediate results) from the optimized DNN. Then the maximal memory size Mem_Part_p of part p could be computed based on the extracted information as:

$$(Mem\_part_p)_{\max} = Mem\_Intr_{i-1} + \sum_{u=i}^{j} Mem\_layer_u + Mem\_Intr_i, \tag{6}$$

where Mem_Layer_u is the memory size of layer u, Mem_Intr_{i-1} and Mem_Intr_i is the memory size of the intermediate result of layer $i-1$ and i, $i \in [1, l]$, $j \in [i+1, l]$, l is the number of layers. When $i = 1$, Mem_Intr_0 is the input sample. Mem_Part_p should also satisfy for constraint of the secure memory as:

$$(Mem\_part_p)_{\max} \leq Mem_{TA}. \tag{7}$$

Based on Equation 6 and Equation 7, we can compute memory size for Mem_Part_p for each part.

Sub-layer Partitioning: If the memory size of a single layer still exceeds the available memory of mobile TEE, the DNN model will fail to execute. To overcome this issue, we propose a sub-layer partitioning method that divides a layer into n sub-layers. Each sub-layer contains a part of neurons and corresponding intermediate results that could be loaded into secure memory safely. Specifically, we partition a convolutional layer into multiple sub-layers, which contain part of the filters and the corresponding output. Furthermore, a fully connected layer is partitioned by multiple sub-layers

by grouping parts of the nodes and their output together. Figure 5 shows an example of proposed sub-layer partitioning methods for the convolutional layer. The filters in one layer are divided into two subsets, which, in turns, produces a subset of the intermediate feature maps. By executing each sub-layer in pieces and combining the intermediate results for the output of the whole layer, the large convolutional layers could be run on the mobile TEE.

The corresponding memory size Mem_Subl_s of each sub-layers can be estimated as:

$$(Mem\_Subl_s)_{\max} = \sum_{v=i}^{j} Mem\_SubIntr_v + Mem\_Neuron_v, \quad (8)$$

where Mem_Neuron_v is the memory size of neuron v, $Mem\_SubIntr_v$ is the memory size of the input and the intermediate result of each sub layer v, $i \in [1, l_s]$, $j \in [i+1, l_s]$, l_s is the number of neurons of each layer. Mem_Subl_s should also satisfy for the secure memory requirments as:

$$(Mem\_Subl_s)_{\max} \leq Mem_{TA}. \quad (9)$$

Next, we can compute memory size Mem_Subl_s for each sub-layer s. After computing the Mem_Part_p and Mem_Subl_s, we partition the optimized DNN model into n parts and save them into separated parts for further encryption and loading. Each part may contain multiple layers or sub-layers of the DNN model and can be loaded into mobile TEE safely.

Multi-layers Loading. After an optimized DNN is partitioned into multiple parts, we extract the partitioning information (i.e., partition point) and generate a multi-layer loading schedule. The partitioned DNN and loading schedule are deployed on mobile devices (e.g., solid-state disk drive (SSD)). Then, the mobile TEE can load the needed part sequence and inference the predictive result based on the loading schedule.

Towards this end, we develop a client application (CA) and a trust application (TA) together to complete multi-layers loading and inference of the DNN on mobile TEE. Once the inference starts, the CA first invokes the TA, which initializes the running environment of the mobile TEE and allocates the secure memory. After the TA runs in mobile TEE, it loads the needed part of the optimized DNN into the secure memory. The TA starts the inference and keeps its output as input for the next partition in the TEE. As each part depends only on the output of the previous part (or input) and is stored separately, it can be loaded by the TA into the secure memory separately. Specifically, the first part with an input sample is passed and executed in secure memory to compute the activations of all the neurons it contains first. Once complete, the neurons of the current part could be discarded and replaced with the next needed part. The current intermediate result is stored in secure memory for use as input to the next part. This process is repeated until all parts of the optimized DNN model are executed in TEE.

Compared with the existing loading methods that load one layer each time, our multi-layers loading approach significantly reduces the loading iterations and time overhead. More importantly, while the existing approaches only focus on partitioning and loading optimizations on DNNs, our approach optimizes both inference and loading efficiency simultaneously since the proposed progressive pruning significantly reduces computation and memory cost.

Table 1: DNN models specifications (COVN: convolutional layer, FC: fully-connected layer)

Model	Dataset	Memory Size (≈ MB)	Architecture
MobileNet-V1	CIFAR-10	5	26 COVNs, 1 FCs
	ImageNet	20	26 COVNs, 3 FCs
GoogLeNet	ImageNet	32	57 COVNs, 1 FCs
AlexNet	ImageNet	320	5 COVNs, 3 FCs
ResNet-50	ImageNet	150	53 COVNs, 1 FCs
VGG-16	CIFAR-10	95	13 COVNs, 2 FCs
	ImageNet	630	13 COVNs, 3 FCs

4 EVALUATION

4.1 Experimental Setup

Implementation. 1) *Server Side*: The training, pruning, and optimization algorithms are implemented using Darknet library written in C and are run on a server. 2) *Mobile Side*: We leverage the DarkneTZ to implement TA and CA on a mobile TEE with OP-TEE operating system, and deploy them on Raspberry Pi 3 Model B+ board (secure memory size is 16MB: 14MB for the TA + 2MB for TEE run-time). Although we prototype on an ARM Cortex-A processor, our approach is portable to lower-end ARM TrustZone processors (e.g., Cortex-M23/33 with 64KB-1MB memory).

Models and Datasets. We evaluate the proposed approach with 5 typical DNNs including 2 small size models (i.e., GoogLeNet [53] and MobileNet [17]) and 3 deeper & larger size models (i.e., AlexNet [23], ResNet-50 [52] and VGG-16 [51]). We run these models on two types of datasets: a smaller one (i.e., CIFAR-10 [22]) and a larger one (i.e., ImageNet [47]). Table 1 summarizes the detailed configurations of models and datasets.

Pruning Setup. We implement a script that utilizes the Random Search [1] to auto-search optimal parameters of the proposed progressive pruning one by one. The optimal parameters are chosen based on which ones can get the highest pruning rate with the least accuracy loss. Note we only need to perform the pruning and optimization process once for each DNN on the server before deploying it into different mobile TEE. So, such a process would not involve much more effort. For each dataset, we derive a parameter setting as $\{\alpha_{k,amp}, \alpha_{k,sup}, \beta, a, b\}$. The range of each parameter is set to $\alpha_{k,amp} \in [1.05, 1.2]$, $\alpha_{k,sup} \in [0.35, 0.95]$, $\beta \in [0.2, 0.7]$, $a \in [0.01, 0.3] * k$, $b \in [0.1, 0.7] * k$, where k is the number of neurons of the target DNN.

1) *CIFAR-10*: parameter settings for MobileNet-V1 and VGG-16 is $\{1.1, 0.6, 0.35, 0.01, 0.1\}$ and $\{1.1, 0.6, 0.25, 0.01, 0.15\}$, respectively. We train DNNs with the SGD with mini-batch size 128, weight decay 0.0005 and momentum 0.9. The training is started with a learning rate 0.1, and decayed by 0.9 at every 20 epochs, with total 150 epochs. 2) *ImageNet*: parameter settings for MobileNet V1, GoogLeNet, ResNet-50, AlexNet, and VGG-16 are $\{1.1, 0.75, 0.5, 0.1, 0.3, \}$, $\{1.1, 0.75, 0.4, 0.1, 0.3\}$, $\{1.1, 0.3, 0.5, 0.05, 0.4\}$, $\{1.1, 0.3, 0.35, 0.05, 0.4\}$ and $\{1.1, 0.65, 0.35, 0.05, 0.3\}$, respectively. With a mini-batch size of 64, the learning rate starts at 0.01, which decays by 0.9 at every 30 epochs. The total epochs are 300.

Baselines. 1) **REE** which runs the entire DNNs on the mobile REE; 2) **Hybrid** which runs the first three layers [6, 36] of the DNNs on mobile TEE, with the remaining layers running in the model REE; 3) **LL-TEE** which runs the entire DNNs layer by layer [35] on the mobile TEE. We use **MUL-TEE** to denote the proposed loading

Table 2: Comparison of different pruning methods on the CIFAR-10 dataset.

	Method	Weight CR(%)	FLOPs CR(%)	Re-iters	ΔAcc(%)
VGG-16 MobileNet-V1	1× baseline	0.0	0.0	150K	0.0
	0.50× baseline	74.9	74.7	150K	-1.2
	DCP [61]	58.0	73.8	300K	+0.41
	Our	**89.2**	**90.2**	**45K**	**+0.3**
	VCNN [60]	73.3	39.1	70K	-0.1
	GAL [27]	77.6	39.6	140K	-0.2
	ASP [28]	92.7	67.0	140K	-0.6
	PFS [54]	48.3	50.0	300K	+0.19
	Our	**88.2**	**87.8**	**60K**	**-0.42**

Table 3: Comparison of different pruning methods on the ImageNet dataset.

	Method	Weight CR(%)	FLOPs CR(%)	Re-iters	ΔAcc(%)
MobileNet-V1	1× baseline	0.0	0.0	150K	0.0
	0.75× baseline	26.0	42.8	150K	-3.8
	AMC [13]	42.8	50.1	300K	+0.41
	PFC [5]	52.5	56.7	300K	+0.41
	Our	**55.2**	**52.3**	**85K**	**-0.4**
GooL	TUKER	69.2	49.5	-	-0.24
	NISP [57]	67.7	42.32	-	-0.23
	Our	**62.1**	**40.1**	**45K**	**-0.32**
ResNet-50	NISP	46.4	41.7	190K	-1.5
	GAL	53.4	50.2	230K	-0.6
	PFC	54.0	50.3	150K	-1.2
	Our	**42.1**	**38.3**	**120K**	**-0.3**
AlexNet	DCP	94.3	-	190K	-0.4
	PFC	95.4	-	120K	-0.2
	SA [54]	95.4	-	100K	-0.3
	Our	**90.1**	**83.5**	**65K**	**-0.4**
VGG-16	DDS	76.8	67.1	210K	-2.0
	AMC	75.0	65.2	260K	-1.4
	SA	73.2	65.0	350K	-0.6
	Our	**74.8**	**62.5**	**150K**	**-0.5**

approach that loads as many layers as possible into mobile TEE on unpruned DNN models.

Evaluation Metrics. We measure the proposed progressive pruning in terms of *FLOPs [8] compression rate (CR)*, *weight CR*, *accuracy drop (ΔAcc)* and *re-train iterations (Re-iters)*. We also measure the *loading iterations* (number of loading times from REE into TEE) and *inference latency* (total time to execute a DNN on user input to generate the output).

4.2 Performance of Pruning

4.2.1 ***Results on CIFAR-10.*** We first conduct the experiments with MobileNet-V1 (width multipliers: 1, 0.75, 0.5, 025) and VGG-16 on CIFAR-10 datasets. The multiplier is used to control the number of channels for all convolution layers. Table 2 shows the comparison results between our pruning methods and state-of-the-art structured pruning methods. Against MobileNet-V1 with the DCP, our approach has 30% and 20% higher Weight CR and FLOPs CR than the 0.5 baseline MobileNet-V1 and DCP, respectively. Compared to existing pruning methods on VGG-16, our approach achieves the comparable weight CR and FLOPs CR, but with 1-5 times fewer retrain iterations. These results validate the effectiveness of our approach on a smaller dataset on both light and large DNN models.

Table 4: Comparison of inference time on the CIFAR-10 dataset.

	Weight CR(%)	Loading Method (Loading Iterations)	Inference Speed(s)	ΔAcc (%)
MobilNet-V1	Baseline (1/0.75/0.5)	REE (N/A)	0.75/0.69/0.63	0/2.7/20.2
		Hybrid (N/A)	4.62/4.21/3.95	
		LL-TEE (27/27/27)	56.72/56.03/55.56	
		MUL-TEE (1/1/1)	**(4.92/4.56/4.12)**	
	60.2% (Progressive pruning)	REE (N/A)	0.58	0.4
		Hybrid (N/A)	3.53	
		LL-TEE (27)	55.12	
		Our (1)	**3.91**	
VGG-16	Unpruned	REE (N/A)	1.52	0
		Hybrid (N/A)	5.53	
		LL-TEE (18)	43.35	
		MUL-TEE (7)	**13.64**	
	88.2% (Progressive pruning)	REE (N/A)	1.06	0.2
		Hybrid (N/A)	5.09	
		LL-TEE (15)	39.23	
		Our (1)	**4.93**	

4.2.2 ***Results on ImageNet.*** To evaluate the effectiveness of our pruning approach on large-scale datasets, we further conduct experiments to prune MobileNet-V1, GoogLeNet, ResNet-50, AlexNet and VGG-16 on the ImageNet dataset. We compare our method with state-of-the-art pruning methods in Table 3. We observe that our pruning approach achieves comparable weight CR and FLOPs CR with fewer retrain iterations than other pruning works. For example, the results show that our approach has 14% and 10% higher weight CR and FLOPs CR than the 0.75 baseline MobileNet-V1. In addition, the results of ResNet-50, AlexNet, VGG-16, and GoogLeNt show that our approach achieves comparable weight CR and higher Flops CR in comparison with other pruning approaches with fewer retrain iterations.

The evaluation results demonstrate that our pruning method can identify and remove the redundant neurons effectively, suggesting that the proposed pruning method can achieve a comparable compression ratio with existing state-of-the-art ones with fewer retrain iterations. More importantly, as our pruning method does not impact the inference accuracy, the various size of DNNs could be compressed and deployed on mobile devices with efficient inference.

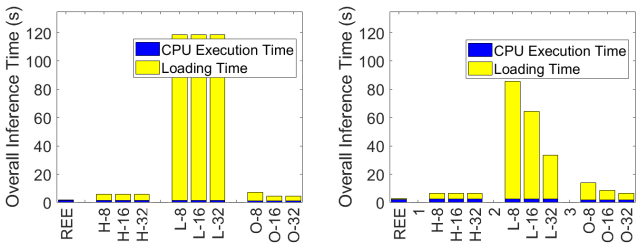
4.3 Inference Latency

4.3.1 ***Results on CIFAR-10.*** Table 4 shows that our approach significantly reduces the loading iterations and improves the inference speed compared to the unpruned/pruned DNNs in 3 baselines. For instance, the loading iterations and inference time of our approach on pruned MobileNet-V1 are 1 and 3.91s, which have $27x/27x$ and $18x/17x$ improvements over the unpruned/pruned LL-TEE baseline, respectively. For VGG-16, we achieve $2.8x/15x$ and $9.8/9.5x$ less loading iterations and inference time over the unpruned/pruned LL-TEE baseline. We also observe that our approach achieves comparable inference time compared with REE and Hybrid baseline on the unpruned MobileNet-V1 and VGG-16 models without the concerns of security and network issues. These results demonstrate the effectiveness of the pruning approach for compressing the DNN models and improving the inference speed.

Table 5: Comparison of inference time on the ImageNet dataset.

	Weight CR(%)	Loading Method (Loading Iterations)	Inference Speed(s)	ΔAcc (%)
MobilNet V-1	Baseline (1/0.75/0.5)	REE (N/A)	1.73/1.56/1.41	0/3.8/8.1
		Hybrid (N/A)	5.22/4.95/4.71	
		LL-TEE (29/29/29)	(62.93/60.61/58.45)	
		MUL-TEE (2/2/1)	**(7.43/7.15/4.35)**	
	52.3% (Progressive pruning)	REE (N/A)	1.34	0.4
		Hybrid (N/A)	4.82	
		LL-TEE (29)	56.92	
		Our (1)	**4.21**	
GoogLeNet	Unpruned	REE (N/A)	1.95	0
		Hybrid (N/A)	5.87	
		LL-TEE (58)	118.34	
		MUL-TEE (5)	**12.52**	
	62.1% (Progressive pruning)	REE	1.46	0.32
		Hybrid	5.29	
		LL-TEE (58)	117.63	
		Our (1)	**4.34**	
AlexNet	Unpruned	REE (N/A)	2.71	0
		Hybrid (N/A)	6.32	
		LL-TEE (23)	64.85	
		MUL-TEE (19)	**40.32**	
	90.1% (Progressive pruning)	REE (N/A)	2.09	0.4
		Hybrid (N/A)	5.85	
		LL-TEE (8)	16.9	
		Our (3)	**8.85**	
ResNet-50	Unpruned	REE (N/A)	2.28	0
		Hybrid (N/A)	6.04	
		LL-TEE (54)	112.2	
		MUL-TEE (11)	**25.33**	
	42.3% (Progressive pruning)	REE (N/A)	1.89	0.31
		Hybrid (N/A)	5.44	
		LL-TEE (54)	110.4	
		Our (7)	**15.23**	
VGG-16	Unpruned	REE (N/A)	Not Support	0
		Hybrid (N/A)	Not Support	
		LL-TEE (49)	Not Support	
		MUL-TEE (43)	**Not Support**	
	75.1% (Progressive pruning)	REE (N/A)	2.64	0.24
		Hybrid (N/A)	6.53	
		LL-TEE (21)	48.73	
		Our (15)	**33.62**	

4.3.2 ***Results on ImageNet.*** We also test our approach on VGG-16, AlexNet, ResNet-50, GooGleNet, and MobileNet-V1 on the ImageNet to study the performance of our approach on a large and complex dataset. Note that we are not able to run the unpruned VGG-16 due to their high memory requirements, which further justifies our idea of using our approach. The results are shown in Table 5. We observe that our approach significantly reduces the loading iterations and improves the inference speed compared to the unpruned/pruned models in 3 baselines. For instance, the inference time of our approach on pruned GoogLeNet is 4.34s, which are 1.3x/1.2x and 27.3x/26.9x improvements over the unpruned/pruned GoogLeNet model in Hybrid and LL-TEE, respectively. For AlexNet, we achieve a 7.4x/1.9x less inference time than the unpruned/pruned LL-TEE baseline. We also observe that we



(a) GoogLeNeton ImageNet **(b) AlexNet on ImageNet**

Figure 6: Performance comparison with different TEE sizes (H: Hybrid, L: LL-TEE, O: Our , 8: 8MB, 16: 16MB, 32: 32MB).

achieve comparable inference time compared with Hybrid baseline on the unpruned/pruned AlexNet model without the concerns of security and network issues. We are aware that the loading iterations of some DNNs exceed the number of layers. For example, the loading iterations of pruned VGG-16 on the ImageNet dataset in LL-TEE and our approach are 21 and 15, which are more than the number of layers 13. The reason is that if the size of one layer is larger than the available size of the mobile TEE, we have to divide them into multiple parts to ensure each part can be executed in the mobile TEE safely. For example, the size of the first fully connected layer of VGG-16 is over 70MB, which is large than the available size of mobile TEE (i.e., 16MB). Our sub-layer partitioning method divides this layer into 6 sub-layers to ensure each sub-layer fits the memory size of the mobile TEE. The evaluation results on CIFAR-10 and ImageNet demonstrate that our approach improves the inference efficiency on various sizes of DNNs models by compressing DNN size and optimizing loading iterations jointly.

4.3.3 ***Ablation Evaluation.***

Impact of Progressive Pruning. We conduct the experiment to gain insight on the effects of the pruning method on the overall performance. The results are shown in Table 4 and Table 5. We observe that the proposed progressive pruning approach significantly reduces the loading iterations and inference time in 3 baselines and MUL-TEE for both 5 models with 2 datasets. Specifically, the inference time is 0.58s, 3.53s, 55.12s, and 3.91s for the pruned MobileNet-V1 model on CIFAR-10, which reduces 40.2%, 10.3%, 2.3%, and 19.2% compared to the unpruned model in REE, Hybrid, LL-TEE, and MUL-TEE. For VGG-16 on CIFAR-10, the pruning approach reduces the inference times by around 43.2%, 17.1%, 6.1%, and 63.2% compared to the unpruned model in REE, Hybrid, LL-TEE, and MUL-TEE. Moreover, the inference time for pruned GoogLeNet in REE, Hybrid, LL-TEE, and MUL-TEE are 0.479s, 0.562s, 13.031s, and 0.452s, respectively, where has 26.3%, 21.1%, 5.4%, and 52.3% improvements over the unpruned model. We also observe the same trend for the other DNN models when using pruning approach. The result demonstrates the effectiveness of the pruning approach for compressing the DNN models and improving the inference speed.

Impact of MUL-TEE Approach. We also study the impact of the multi-layers loading method by experimenting with unpruned/pruned DNN models compared with the LL-TEE baseline. For instance, for the unpruned/pruned VGG-16 model on the CIFAR-10 dataset, MUL-TEE achieves 2.5x/15x and 3.4x/9.3x less loading iterations and inference time than the LL-TEE baseline. Moreover, the inference time of unpruned/pruned GoogLeNet, AlexNet, and ResNet-50 of MUL-TEE are 12.52s/4.34s, 40.32s/8.85s,

Table 6: Comparison of inference time overhead with existing works that execute DNNs on mobile TEE. (Note *: Entire DNN in Mobile and Cloud TEE, †: A first few layers in Mobile TEE, §: Only last layer in Mobile TEE,**: Entire DNN in Mobile TEE, NS: Not Support,NR: Not Report).

Work	ResNet-50	AlexNet	VGG-16	GoogLeNet
HybridTEE [41]	NR	NR	NR	6.2×*
DarkneTZ [36]	-0.8×†	-5.1×§	NS	NS
TrustedDNN [30]	NS	-5.9×**	NS	NS
Our	**-6.8**×**	**-3.7**×**	**-11.2**×**	**-3.2**×**

and 25.33s/15.23s, which are $9.2x/29.4x$, $1.2x/2.2x$, and $5.2x/6.9x$ faster than LL-TEE baseline. The result confirms the effectiveness of the multi-layers loading method in improving the inference efficiency compared to existing TEE approaches.

Impact of TEE Memory Size. In order to evaluate the impact of TEE size on system performance, we examine the overall inference time of our approach for GoogLeNet and AlexNet with REE, Hybrid, and LL-TEE baseline using different TEE sizes (i.e., 8MB, 16MB, and 32MB). Note that 32MB of TEE size is not a good practice since it increases the level of exploitable security vulnerabilities [30]. We decompose the overall inference time into two parts: 1) *CPU Execution Time* which measures the time spent on computing the DNN inference; 2) *Loading Time* which measures the time on loading needed parts into mobile TEE for inference. Figure 6 shows the results, which indicate that inferences on mobile TEE introduce more time overhead compared with REE baseline.

We observe that the loading process introduces most of the time overhead compared with execution time. Thus, optimizing the loading time is a critical process for efficient inference DNNs on mobile TEE. The results show we can significantly reduce the loading time by 2-4 times on two DNNs with the increasing of TEE sizes. In contrast, the Hybrid and LL-TEE cannot reduce the loading time for GoogLeNet with the increasing of TEE sizes since the GoogLeNet consists of 58 layers, which need to be loaded many times. Furthermore, the results show our approach has 60% and 85% less CPU execution time compared with REE, Hybrid, and LL-TEE baseline for GoogLeNet and AlexNet, respectively. The insight behind the observation is that we could load more or all layers of the pruned DNN into TEE in one iteration. The results demonstrate that our approach is effective in reducing the inference time overhead on the various DNNs by efficiently utilizing the various TEE sizes.

4.3.4 ***Comparison with Existing Studies.*** We compare the inference time with three similar existing works in terms of mobile REE baseline in Table 6. We noticed that each work uses different hardware and software, which will make the comparison unfair. So, we directly derive the results from their works to demonstrate the performance using the TEE before and after (note that low-end devices would generate more improvement than high-end devices). The results demonstrate that our works are able to deploy the large-size DNNs on resource-constraint mobile TEE by leveraging the proposed progressive pruning approach. While other approaches only deploy a few large-size DNNs on mobile TEE. We can see from Table 6 that our approach has the 3.7x inference time overhead on AlexNet, which outcomes the TrustedDNN and DarknetTZ. The HybridTEE achieves 6.2x inference time improvements compared to the REE baseline on GoogLeNet. That is because they run most layers of DNNs in the cloud server to utilize its powerful computing resources. Although our approach has more inference time overhead, the entire DNN is protected by local mobile TEE without the concern of network connection issues. Moreover, our approach introduces 6.8x inference time overhead on ResNet-50, which is greater than DarkneTZ (i.e., -0.8x). However, DarkneTZ only protects the last layer by mobile TEE, which will increase the risks of adversary attacks. The results demonstrate that our approach can deploy the various size of DNNs on mobile TEE with a fewer inference time overhead compared with existing approaches.

4.4 Discussion

1) Security and Privacy Analysis. In this paper, the entire DNN inference process is protected by the hardware-based mobile TEE, ensuring that attackers cannot access the DNN weight and intermedia result to compromise the DNN inference. Therefore, our approach is capable of defending MIA, DRA, PIA, and GAN [15] attacks. We can also eliminate layers' information exposure to attacks (e.g., Bit-flip attacks) to recover images and texts from intermediate gradients. Moreover, we have found that side-channel attacks on CPU cache [12] could compromise the integrity of DNNs. We plan to explore the side-channel attack protection methods (e.g., Privado [9]) to protect DNN data inside the mobile TEE. **2) Supported Machine Learning Models.** In this paper, we focus on typical used DNNs, including convolution and fully-connected layers. We plan to apply our approach to other Machine Learning models (e.g., recurrent neural networks). In addition, we could also take the state-of-art compression approaches with quantization that achieve a higher compression rate to apply the more large size of DNNs on mobile TEE. **3) Training and Pruning on Mobile TEE.** Currently, our work focuses on DNNs inference. We have found that it is difficult to train or prune a DNN model inside mobile TEE since they demand significant computational and memory resources compared to inference. We plan to explore efficient methods for training and pruning on the mobile TEE (e.g., tiny batch size training [31]). **4) Efficient Decryption on Mobile TEE.** In this paper, the pruned DNN model and input are assumed encrypted and stored in REE. We are aware that the decryption process will introduce inference time overhead. However, existing study [30] demonstrate that the decryption process is practical to use and only introduces 3%-5% time overhead of the inference time. We plan to adopt efficient encryption/decryption algorithms (e.g., OHE [20]) to protect the DNN on mobile REE and decrypt it into mobile TEE. **5) Power Consumption Measurements.** We estimate the rough energy consumption for our approach compared with LL-TEE baselines by using the FLOPs and loading time reduction, which are linearly proportional to the energy consumption [29]. Since our approach can achieve 2-10 times FLOPs and 8-30 times loading time reduction, we could reduce the energy consumption by around 2-10 times and 2-30 times for CPU execution and loading processes, respectively. To accurately measure the power consumption, we plan to use an off-the-shelf high-precision power monitor to track the power consumed by a mobile device.

5 RELATED WORK

5.1 Optimizing DNN Inference

Pruning is one of the most popular network optimization techniques due to its effectiveness in reducing network complexity. Although

structured pruning achieves a relatively lower pruning rate than unstructured pruning, it does not require specific hardware platforms to obtain acceleration in practice. For example, Zhao *et al.* [60] re-formulate the channel pruning problem within Bayesian probabilistic learning to operate on a redefined scaling factor in Batch Normalization. Lin *et al.* [26] globally prune the unimportant filters across all layers first, then recover the mistakenly pruned filters to improve the accuracy. Different from the existing approaches to identifying the redundant parameters coarsely, we compute and adjust the importance of the parameters gradually, resulting in accurate pruning and reduced retraining iterations.

Furthermore, existing pruning approaches (e.g., [60, 61]) have adopted the static memory allocation technique (e.g., tensor or array in PyTorch) for the parameters of DNNs. As a result, the pruned parameters still occupy the memory space and are involved in the computation process of inference. To address the issues, researchers developed software and hardware co-design to eliminate redundant computation and memory costs. For example, EIE [11] proposed an inference engine to skip the multiplication of the pruned parameters by designing a scalable array of processing elements in memory. ISAAC [48] reduced the multiply-accumulate operations involved in DNNs by using the memristor crossbar arrays. Although such works could achieve inference improvements on pruned DNNs, they need dedicated hardware designs, which are not practical for some mobile devices. In contrast, our work optimizes the pruned DNNs by removing the pruned memory occupation by only using low-level programming techniques. Thus, our approach can easily be applied to mobile devices at a low cost.

5.2 Preserving Privacy of DNN Models

Researchers have found that attackers can leverage the memorized information of DNN layers (e.g., memory content and address) to infer sensitive properties about the input data, leading to severe privacy risks [43]. In addition, the confidentiality and integrity of a DNN model itself are subject to a variety of security threats [33]. Therefore, various security methods have been proposed to defend against these threats (e.g., k-anonymity [25], randomized noise addition [37], and encryption-based [42]). However, such methods are hard to be applied to protect DNNs on mobile devices since they involve intensive computational costs.

Recently, researchers have adopted TEE to protect DNN since it provides better computation efficiency. However, they still have to overcome the resource constraint of TEE on mobile devices. Along this direction, researchers have developed several TEE-oriented partitioning techniques [10, 36] to partition DNN and run the part of it inside the TEE on the mobile device or cloud server, respectively. For example, DarkneTZ [36] runs the first few layers and the last layer inside the TEE on mobile devices and leaves the other layers unprotected to defend against MIA. However, attackers could still perform DRA and PIA attacks on unprotected parts. To protect all layers of DNNs, HybridTEE [6] places the first few layers in the TEE on mobile devices, while other layers are offloaded to the TEE on the cloud servers. Trusted-DNN [30] partitions the convolutional matrix into multiple small ones and leverages the quantification method (32 bit to 1 bit) to compress the DNN to reduce the memory demand. However, such an approach suffers a significant accuracy drop.

Unlike these works that only focus on the partition of the DNN, we first prune the DNN to reduce computational and memory costs while maintaining high accuracy. Then, the pruned model is optimized for real-world acceleration and partitioned into multiple parts containing as many layers as possible while fitting the memory constraint of the mobile TEE. Therefore, our approach promises entire DNNs to be executed in the mobile TEE with minimal inference time overhead and without network connection issues.

6 CONCLUSION

In this work, we develop a novel mobile TEE-based security framework to securely execute entire DNNs in the resource-constrained mobile TEE only introducing a few inference time overheads. Specifically, we first gradually prune the redundant neurons to reduce the memory and computational cost in a fine-grained way. Then, we deallocate the memory storage of pruned neurons in a real implementation by utilizing the low-level programming technique. To realize a minimized loading time overhead, we develop an adaptive partition mechanism that partitions as many layers as possible into one part while fitting the memory constraint of mobile TEE. Our experiments with various DNNs and open-source datasets demonstrate that we can achieve 2-30 times less inference time with comparable accuracy compared to existing approaches securing entire DNNs with mobile TEE.

ACKNOWLEDGMENT

This work was partially supported by the the National Science Foundation Grants CCF1909963, CCF2211163, CNS2120396, CNS2304766, CNS2145389, CNS2120276, CCF2000480, CCF2028873, and CNS2120350.

REFERENCES

[1] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* 13, 1 (2012), 281–305.
[2] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1416–1432.
[3] Victor Costan and Srinivas Devadas. 2016. Intel sgx explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.
[4] Scott G Dacko. 2017. Enabling smart retail settings via mobile augmented reality shopping apps. *Technological forecasting and social change* 124 (2017), 243–256.
[5] Tim Dettmers and Luke Zettlemoyer. 2020. Sparse Networks from Scratch: Faster Training without Losing Performance. https://openreview.net/forum?id=ByeSYa4KPS
[6] Akshay Gangal, Mengmei Ye, and Sheng Wei. 2020. HybridTEE: Secure Mobile DNN Execution Using Hybrid Trusted Execution Environment. In *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 1–6.
[7] Simson Garfinkel, John M Abowd, and Christian Martindale. 2019. Understanding database reconstruction attacks on public data. *Commun. ACM* 62, 3 (2019).
[8] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)* 23, 1 (1991), 5–48.
[9] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and secure DNN inference with enclaves. *arXiv preprint arXiv:1810.00602* (2018).
[10] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Confidential inference via ternary model partitioning. *arXiv preprint arXiv:1807.00969* (2018).
[11] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016).
[12] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, et al. 2018. Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567* (2018).
[13] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[14] Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al. 2019. Streaming end-to-end speech recognition for mobile devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 6381–6385.

[15] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 603–618.

[16] Xueshi Hou, Yao Lu, and Sujit Dey. 2017. Wireless VR/AR with edge/cloud computing. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–8.

[17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[18] Bin Hu and Jiacun Wang. 2020. Deep learning based hand gesture recognition and UAV flight controls. *International Journal of Automation and Computing* 17, 1 (2020), 17–29.

[19] Bin Hu, Tianming Zhao, Yucheng Xie, Yan Wang, Xiaonan Guo, Jerry Cheng, and Yingying Chen. 2021. Details omitted for double-blind reviewing. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[20] G Kalyani and Shilpa Chaudhari. 2020. An efficient approach for enhancing security in Internet of Things using the optimum authentication key. *International Journal of Computers and Applications* 42, 3 (2020), 306–314.

[21] Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. 2020. Vessels: Efficient and scalable deep learning prediction on trusted processors. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 462–476.

[22] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[24] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*.

[25] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2005. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 49–60.

[26] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2018. Accelerating Convolutional Networks via Global & Dynamic Filter Pruning.. In *IJCAI*.

[27] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. 2019. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*.

[28] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2020. AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates.. In *AAAI*. 4876–4883.

[29] Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, and Mani Srivastava. 2021. SecDeep: Secure and Performant On-device Deep Learning Inference Framework for Mobile and IoT Devices. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*. 67–79.

[30] Zhuang Liu, Ye Lu, Xueshuo Xie, Yaozheng Fang, Zhaolong Jian, and Tao Li. 2021. Trusted-DNN: A TrustZone-based Adaptive Isolation Strategy for Deep Neural Networks. In *ACM Turing Award Celebration Conference-China (ACM TURC 2021)*.

[31] Dominic Masters and Carlo Luschi. 2018. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612* (2018).

[32] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N Asokan. 2015. Open-TEE–An Open Virtual Trusted Execution Environment. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 400–407.

[33] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 691–706.

[34] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. 2020. A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954* (2020).

[35] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. *arXiv preprint arXiv:2104.14380* (2021).

[36] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*.

[37] Shibnath Mukherjee, Zhiyuan Chen, and Aryya Gangopadhyay. 2006. A privacy-preserving technique for Euclidean distance-based mining algorithms using Fourier-related transforms. *The VLDB journal* 15, 4 (2006), 293–315.

[38] Ben Mussay, Margarita Osadchy, Vladimir Braverman, Samson Zhou, and Dan Feldman. 2019. Data-independent neural pruning via coresets. In *International Conference on Learning Representations*.

[39] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. 113–124.

[40] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 907–922.

[41] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2020. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal* 7, 5 (2020), 4505–4518.

[42] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big data analytics over encrypted datasets with seabed. In *12th USENIX symposium on operating systems design and implementation*.

[43] Adityanarayanan Radhakrishnan, Caroline Uhler, and Mikhail Belkin. 2018. Downsampling leads to Image Memorization in Convolutional Autoencoders. (2018).

[44] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1211–1220.

[45] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[46] Ronald Rivest. 1992. *The MD5 message-digest algorithm*. Technical Report.

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[48] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[49] Yonghee Shin and Laurie Williams. 2008. An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*.

[50] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[51] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[52] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[54] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. 2020. Pruning from Scratch.. In *AAAI*. 12273–12280.

[55] Mengmei Ye, Jonathan Sherman, Witawas Srisa-An, and Sheng Wei. 2018. TZSlicer: Security-aware dynamic program slicing for hardware isolation. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*.

[56] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[57] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*.

[58] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 365–382.

[59] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*.

[60] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. 2019. Variational convolutional neural network pruning. In *CVPR*.

[61] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. 2018. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*.