SelTZ: Fine-Grained Data Protection for Edge Neural Networks Using Selective TrustZone Execution

Sehyeon Jeong and Hyunyoung Oh *

Posted Date: 25 November 2024

doi: 10.20944/preprints202411.1852.v1

Keywords: Edge; IoT; TrustZone; Membership Inference Attack; Deep Learning

 doi:10.20944/preprints202411.1852.v1

Article

SelTZ: Fine-Grained Data Protection for Edge Neural Networks Using Selective TrustZone Execution

Sehyeon Jeong and Hyunyoung Oh *

Department of AI·Software, Gachon University, Seongnam-si 13120, Gyeonggi-do, Republic of Korea
* Correspondence: hyoh@gachon.ac.kr

Abstract: This paper presents an approach to protecting deep neural network privacy on edge devices using ARM TrustZone. We propose a selective layer protection technique that balances performance and privacy. Rather than executing entire layers within the TrustZone secure environment, which leads to significant performance and memory overhead, we selectively protect only the most sensitive subset of data from each layer. Our method strategically partitions layer computations between normal and secure worlds, optimizing TrustZone usage while providing robust defenses against privacy attacks. Through extensive experiments on standard datasets (CIFAR-100 and ImageNet-Tiny), we demonstrate that our approach reduces membership inference attack (MIA) success rates from over 90% to near random guess (50%) while achieving up to 7.3× speedup and 71% memory reduction compared to state-of-the-art approaches. On resource-constrained edge devices with limited secure memory, our selective approach enables protection of significantly more layers than full layer protection methods, while maintaining strong privacy guarantees through efficient data partitioning and parallel processing across security boundaries.

Keywords: Edge, IoT, TrustZone, Membership Inference Attack, Deep Learning

1. Introduction

The rapid growth of edge computing and Internet of Things (IoT) devices has intensified the demand for on-device machine learning, particularly for deep learning (DL) inference tasks [1]. Edge-based inference offers advantages in latency, data privacy, and reduced dependency on network connectivity. However, it also presents substantial challenges in safeguarding model data and user information, particularly on untrusted edge devices. Recent studies have shown that deep neural networks (DNNs) are vulnerable to privacy attacks, including *membership inference attacks* (MIAs) [2], where adversaries can identify if specific data points were part of the model's training set. These vulnerabilities pose serious privacy risks in sensitive domains such as healthcare and finance.

Traditional methods to secure DNNs, such as homomorphic encryption [3] and differential privacy [4], impose high computational costs or reduce model accuracy, making them less suitable for resource-constrained edge devices. *Trusted Execution Environments* (TEEs), specifically ARM TrustZone (TZ) [5], have emerged as viable hardware-based solutions to enhance the security and privacy of edge-based machine learning. TrustZone offers an isolated secure environment, separate from the primary operating system, where sensitive computations can be processed without interference from the normal world. However, the limited memory available in TZ restricts its capacity to protect large DNN models fully, particularly for complex architectures [6].

Previous methods like *DarkneTZ* [7] addressed these limitations by partitioning DNNs into layers and executing only the most sensitive layers within TZ. Typically, DarkneTZ starts from the final layer and moves backward, securing as many layers as the TZ memory allows. This layer-wise approach protects sensitive outputs that are often targeted by MIAs. However, it also exposes several limitations: (1) for larger models, only a subset of layers can be protected within TZ, leaving other parts exposed; and (2) copying entire layers into TZ incurs significant delays due to data transfer and memory constraints.

To address these challenges, we propose *SelTZ*, a selective layer protection mechanism that secures only the most sensitive portions of each layer within TZ rather than entire layers. Instead of

 doi:10.20944/preprints202411.1852.v1

partitioning feature maps, which would require significant memory overhead, SelTZ partitions model parameters between the *normal world* and *secure world* (TZ). This approach leverages the observation that protecting critical parameters can effectively mitigate MIAs while optimizing computational efficiency. By processing sensitive parameters in TZ while handling non-sensitive ones in the normal world, this selective protection reduces memory load in TZ, enables parallel processing across both worlds, and minimizes data-transfer overhead.

The main challenges we address in SelTZ include:

- **Layer Sensitivity Assessment**: Determining which portions of each layer's computations need to be protected to prevent MIAs while maximizing normal world utilization. SelTZ employs a probabilistic selection strategy that focuses on protecting activation outputs and parameters critical for secure computation.
- **Efficient Cross-World Data Management**: Partitioned computations across worlds introduce data transfer and context switching overheads. SelTZ addresses this through shared memory management and multi-threaded execution, enabling parallel processing while maintaining security boundaries.
- **Layer-Specific Processing Strategies**: Different layer types require specialized handling due to their unique computational characteristics. Convolutional and fully connected layers use parameter partitioning with secure combining, while normalization layers require complete secure world execution.

We validate SelTZ through extensive experiments, demonstrating significant reductions in MIA success rates and improved computational performance compared to existing methods. Our results show that our probabilistic parameter selection strategy, combined with efficient shared memory management and specialized layer processing techniques, effectively balances privacy protection and efficiency. This approach achieves robust defense against MIAs while requiring substantially less TZ memory than full layer protection methods. The modular design of SelTZ makes it adaptable to various DNN architectures on edge devices while preserving inference accuracy.

The remainder of this paper is organized as follows: Section 2 provides background on privacy challenges in edge-based deep learning and reviews related work on TrustZone-based protection. Section 3 details our selective protection approach, including sensitivity assessment, efficient cross-world computation, and layer-specific processing strategies. Section 4 presents the implementation details of our approach on different neural network architectures. Section 5 provides comprehensive experimental results comparing SelTZ with DarkneTZ across different architectures and datasets, demonstrating improvements in both privacy protection and resource efficiency. Section 6 concludes with a discussion of future research directions.

2. Background and Related Work

2.1. Privacy Challenges in Edge-Based Deep Learning

As deep learning (DL) models are increasingly deployed in edge computing and Internet of Things (IoT) applications, significant privacy and security concerns have emerged. Edge devices, by performing data-intensive tasks locally, reduce latency and dependence on cloud resources. However, due to their often untrusted environments, they are highly susceptible to various attacks, including *membership inference attacks* (MIAs) [2,8]. MIAs exploit the ability of an adversary to infer if specific data points were used in training a model, potentially exposing sensitive information, especially in applications involving user-specific data such as medical records or financial transactions. Such attacks take advantage of subtle model behaviors that differ between data it has seen and unseen data, creating a serious privacy vulnerability in edge-deployed models.

A range of techniques have been proposed to defend against MIAs and other privacy risks in DL, including *homomorphic encryption* [3], *secure multi-party computation* [9], and *differential privacy* [4]. Homomorphic encryption enables computations on encrypted data, but it remains computationally

 doi:10.20944/preprints202411.1852.v1

prohibitive for resource-limited edge devices. Differential privacy adds noise to model outputs to obscure individual data contributions, yet this can reduce model accuracy and utility. Consequently, such methods are less practical in latency-sensitive and compute-constrained edge settings. To address these constraints, *Trusted Execution Environments (TEEs)*, specifically ARM TrustZone [5], have emerged as a promising hardware-based solution. TEEs provide an isolated, secure processing area that operates separately from the primary system, allowing edge devices to run sensitive computations in a secure space protected from tampering or unauthorized access.

2.2. Existing Approaches to Secure Deep Learning with TrustZone

Among TEE-based solutions, *DarkneTZ* is a significant method that utilizes TrustZone to secure DL models on edge devices [7]. DarkneTZ partitions deep neural networks (DNNs) at the layer level, prioritizing protection of layers from the last layer backward, as final layers often contain data most vulnerable to MIAs[8]. By copying entire layers into TrustZone until the memory limit is reached, DarkneTZ selectively protects as much of the model as TrustZone memory allows. This approach effectively reduces MIA success rates by isolating sensitive layers that contribute most to inference output.

While DarkneTZ addresses TrustZone's memory constraints, it also presents limitations. First, protecting entire layers restricts DarkneTZ's ability to secure large models, as TrustZone's memory can only hold a limited number of layers in full. Consequently, the unprotected layers remain exposed, posing a risk for MIAs and other inference-based attacks [10–12]. Additionally, the method of copying entire layers into TrustZone is associated with significant latency, particularly during data transfers. The overhead from this layer-wise copying and execution in TrustZone can compromise real-time inference speeds, limiting the utility of DarkneTZ in latency-sensitive applications.

2.3. SelTZ: Selective Protection through Fine-Grained Layer Partitioning

To address the limitations in layer-wise protection, we propose *SelTZ*, a fine-grained protection approach that secures only the most sensitive portions of layer data rather than entire layers. This selective protection strategy is based on the hypothesis that partial protection of critical data elements within each layer is sufficient to defend against MIAs while optimizing computational efficiency. SelTZ splits computations within each layer, processing sensitive data in TrustZone and handling non-sensitive data in the normal world, thus enabling multi-threaded, parallel execution across both environments. This design leverages TrustZone's secure capabilities without fully occupying its limited memory, thus maximizing both efficiency and security.

The SelTZ design introduces several key technical innovations that enhance privacy protection and reduce computational overhead:

- **Efficient Data Management and Multi-Threading**: By classifying data as either sensitive or non-sensitive, SelTZ minimizes the need to transfer large amounts of data between the normal and secure worlds. Only essential, sensitive data is stored within TrustZone, while non-sensitive data resides in shared memory accessible to both worlds. This selective data management reduces data copying, enables efficient memory use, and supports parallel processing across worlds, significantly decreasing latency compared to layer-wise copying approaches.
- **Securely Combining Partitioned Computations**: SelTZ incorporates zero-padding into partitioned weights and biases in both convolutional and fully connected layers, allowing partial results from each world to be securely combined in TrustZone via summation. For instance, convolutional layers are split by partitioning weights rather than input data, avoiding the need to track and secure large inputs. Zero-padding facilitates secure summation of intermediate results from each world in TrustZone, ensuring robust privacy with minimal data transfer.
- **Complete Computation of Non-Partitionable Layers**: Certain layers, such as Softmax and other normalization layers, require complete access to their input data for accurate computation and

 doi:10.20944/preprints202411.1852.v1

thus cannot be partitioned across worlds. SelTZ addresses this by processing such layers fully within TrustZone, ensuring that the outputs remain protected from exposure.

SelTZ represents a major shift from traditional TEE-based solutions by allowing for the fine-grained partitioning of layer computations rather than full layer processing within TZ. Unlike DarkneTZ's approach, which is limited by TrustZone's constrained memory, SelTZ dynamically adjusts the level of protection based on data sensitivity within each layer. This selective partitioning enables robust MIA defense with minimal impact on model efficiency, supporting more extensive and complex models on edge devices. Our design demonstrates that a strategic, data-sensitive approach can achieve a balance between privacy protection and computational feasibility on edge hardware.

The next section details the design and implementation of SelTZ, focusing on its partitioning strategy, secure data management, and the computational workflow for parallel processing across TrustZone and the normal world.

3. Design

3.1. Overview of SelTZ's Approach

SelTZ is a fine-grained data protection mechanism for deep neural networks (DNNs) on edge devices using ARM TrustZone (TZ). Unlike prior methods such as DarkneTZ, which protect entire layers within TZ, SelTZ selectively secures only the most sensitive portions of layer data, leaving non-sensitive data to be processed in the normal world. This approach is based on the hypothesis that targeted protection of sensitive portions is sufficient to defend against membership inference attacks (MIAs), thus optimizing the use of TZ's limited resources and enhancing overall processing efficiency.

DarkneTZ secures layers by processing them sequentially from the last layer backward within TZ until memory limits are reached, as final layers generally contain data most vulnerable to MIAs. However, this approach restricts protection capabilities with larger models due to the TZ memory constraints and the substantial overhead associated with copying entire layers between normal and secure worlds. SelTZ instead partitions layer computations across the secure and normal worlds, allowing more efficient processing by protecting only essential data in TZ while keeping less critical data in the normal world. This division reduces the memory load in TZ, supports concurrent processing in both worlds, and minimizes data-transfer overhead.

Figure 1 illustrates the overall architecture and data flow of SelTZ. The system processes neural network layers (e.g., Conv, ReLU, Max-Pool, FC, Softmax) by partitioning their computations across normal and secure worlds. Rather than partitioning input data, which would require duplicating large feature maps, SelTZ partitions model parameters (weights W and biases b) into two portions: one for normal world processing (denoted with subscript n) and another for secure world processing (denoted with subscript s). These partitioned parameters are used to compute partial results in their respective worlds in parallel. The partial results are then combined in the secure world to maintain security. This design ensures that even if an adversary in the normal world has access to the input data, they can only compute partial results using the parameters available in the normal world (W_n, b_n), while the secure parameters (W_s, b_s) remain protected in TrustZone. Additionally, our design employs a probabilistic parameter selection strategy to determine which portions should be protected, balancing security requirements with computational efficiency. Consequently, the adversary cannot reconstruct the complete computation result without access to the secure world parameters, effectively protecting the model's sensitive components. The detailed mathematical formulations and security mechanisms for each operation will be elaborated in the following sections. The figure also shows how SelTZ manages memory through dedicated secure/normal regions and a shared memory, enabling efficient data transfer between the two worlds while maintaining security boundaries.

 doi:10.20944/preprints202411.1852.v1

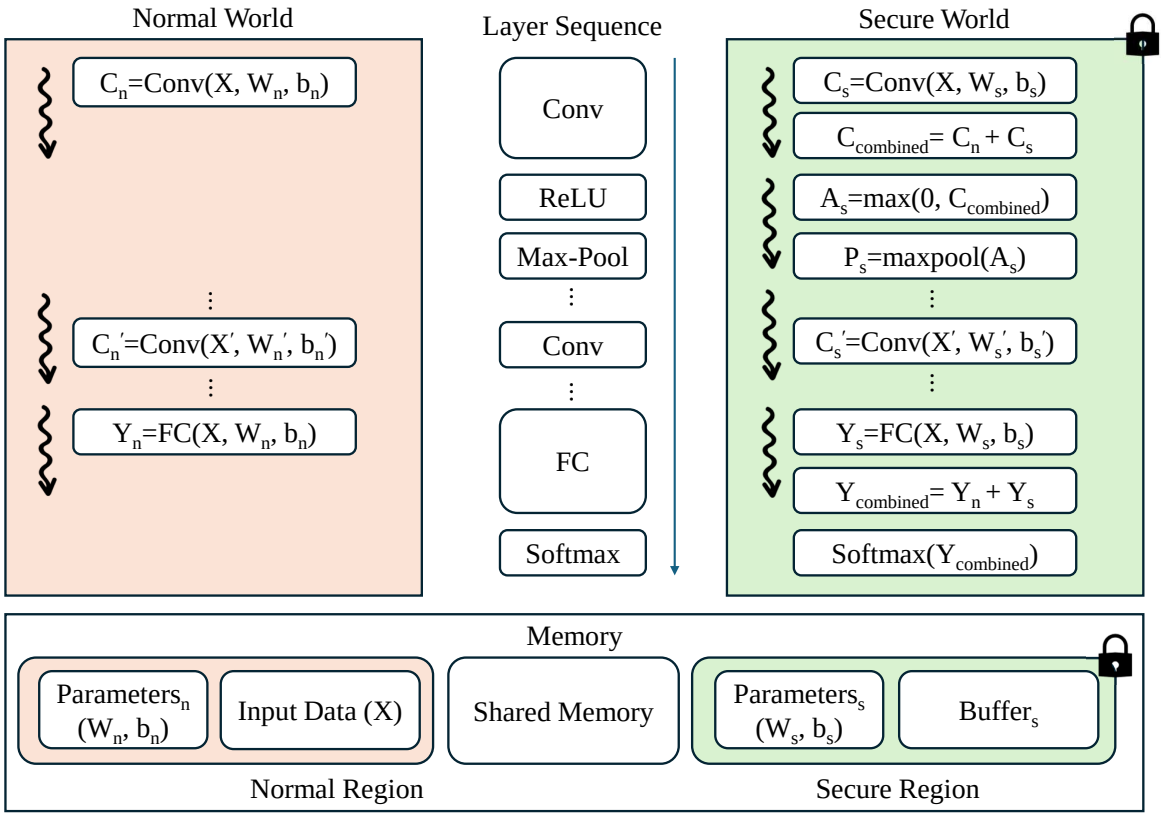


Figure 1. Overview of SelTZ architecture and data flow across normal and secure worlds. The figure shows how different layer types (Conv, ReLU, Max-Pool, FC, Softmax) are processed in parallel across both worlds, with the selective protection mechanism and memory management strategy.

3.2. Challenges and Solutions

The implementation of SelTZ addresses three main challenges. First, we need a systematic approach to assess layer sensitivity and determine optimal partitioning strategies that prevent MIA while maximizing normal world resource utilization. Second, we must efficiently manage data movement and parallel computation across the security boundary while dealing with TrustZone's memory constraints. Third, each layer type (convolutional, fully connected, and normalization layers) requires specialized handling due to their different computational characteristics and security requirements. In this section, we present our solutions through a combination of randomized parameter partitioning, efficient shared memory management, and layer-specific processing strategies.

3.2.1. Layer Sensitivity Assessment and Partitioning Strategy

The primary objective of SelTZ's partitioning strategy is to prevent membership inference attacks (MIAs) by ensuring that attack-critical intermediate outputs remain exclusively in the secure world. Specifically, activation outputs (e.g., ReLU outputs) and final layer outputs, which are commonly exploited in MIAs, must be computed and stored only in TrustZone, preventing any possibility of reconstruction or deduction from the normal world. Beyond these critical components, SelTZ aims to maximize the utilization of normal world resources for other computations to optimize overall performance.

Following this strategy, our partitioning approach first allocates memory for essential secure computations:

- Activation outputs across all layers
- Final layer outputs (e.g., Softmax outputs)
- Associated parameters required for computing these secure outputs

 doi:10.20944/preprints202411.1852.v1

If the remaining TrustZone memory permits, additional computations are selectively moved to the secure world based on their sensitivity scores. For a layer l, we compute its security requirement score R_l as:

$$R_l = \begin{cases} 1.0 & \text{for output of activation or final layer} \\ 1 - \frac{d_l}{D} & \text{for other layers} \end{cases}$$

where d_l is the distance from the output layer and D is the total network depth. This ensures that layers closer to the output receive higher priority when additional secure memory is available, similar to DarkneTZ's backward protection strategy.

Given TrustZone's memory constraint M_{TZ}, the layer selection problem is formulated as:

$$\begin{aligned} \text{maximize} \quad & \sum_l s_l \cdot R_l \cdot m_l \\ \text{subject to} \quad & \sum_l s_l \cdot m_l \leq M_{\text{TZ}} \\ & s_l = 1 \text{ when } R_l = 1.0 \\ & s_l \in \{0, 1\} \quad \forall l \end{aligned}$$

where s_l is a binary selection variable indicating whether layer l is allocated to TrustZone ($s_l = 1$) or not ($s_l = 0$), and m_l is the memory requirement of layer l. The constraint $s_l = 1$ when $R_l = 1.0$ ensures that layers containing critical computations are always allocated to TrustZone. This knapsack-like problem is solved greedily by selecting layers in descending order of their security requirement scores R_l until the memory constraint is reached.

For the layers selected for protection, weights and biases are then partitioned based on their sensitivity scores. For each selected layer l, we define a sensitivity score matrix $S_{i,j}$ using a sensitivity assessment function ψ:

$$S_{i,j} = \psi(W_{i,j}, l)$$

Here, $\psi(W_{i,j}, l)$ is a function that evaluates how much each weight parameter contributes to the layer's output sensitivity, producing values in the range [0,1]. Theoretically, *Shapley* value [13] could be an ideal choice for this assessment, as it precisely quantifies each weight parameter's marginal contribution to the network's outputs by considering all possible parameter combinations. However, computing exact Shapley values requires extensive computational resources and time, making it impractical for real-time applications. Therefore, we propose several efficient yet effective methods to assess parameter importance based on both individual parameter values and their structural relationships within the layer:

1. **Global Importance-based Selection**: Parameters are chosen based on their absolute values, reflecting their direct contribution to the layer's output. For a given protection ratio $\rho_l \in [0, 1]$:

$$\psi(W_{i,j}, l) = \begin{cases} 1 & \text{if } |W_{i,j}| \text{ is among top } \rho_l \text{ fraction of parameters} \\ 0 & \text{otherwise} \end{cases}$$

2. **Structured Pattern Selection**: Recognizing that neural networks often exhibit spatial locality in their feature representations, we propose a systematic approach that considers the collective importance of structurally related parameters. For each pattern type, we first identify all possible pattern instances within the weight tensor and compute their cumulative importance scores based on the absolute sum of their constituent weights:

$$\text{Score}(P) = \sum_{(i,j) \in P} |W_{i,j}|$$

 doi:10.20944/preprints202411.1852.v1

where P represents a specific pattern instance (e.g., a 2×2 block or a row of weights). The patterns are then selected in descending order of their scores until approximately ρ_l fraction of total parameters is covered. This approach ensures that we protect the most influential structural components of the network. Specifically, we consider:

- Row/Column patterns: For each row/column in the weight tensor, compute the absolute sum of its weights. Select rows/columns in descending order of these sums until reaching the target ratio.
- Block patterns (2×2, 3×3): For each possible block position in the weight tensor, compute the absolute sum of weights within that block. Blocks with higher cumulative absolute values are prioritized for protection.

For any selected pattern instance P, the sensitivity function assigns:

$$\psi(W_{i,j}, l) = \begin{cases} 1 & \text{if } (i,j) \in P \text{ and } P \text{ is among selected patterns} \\ 0 & \text{otherwise} \end{cases}$$

This approach of partitioning weights, rather than input features, offers several significant advantages. First, it is more memory-efficient as weight parameters typically require less storage than intermediate feature maps. Second, and more importantly, it provides a novel and highly effective protection mechanism: even hiding a small fraction of critical weights can influence the entire feature map computation through the partial sum operations. This is because each weight parameter contributes to multiple elements in the output feature map through convolution or matrix multiplication operations. Consequently, protecting a carefully selected ρ_l fraction of weights can effectively "spread" the protection across the entire feature space, achieving substantial security benefits with minimal secure memory usage. **This multiplicative effect of weight protection represents a key advantage in our approach, enabling robust defense against MIAs while maintaining efficient resource utilization**. Additionally, since weights remain constant during inference, this approach simplifies the management of sensitive data across world boundaries.

3.2.2. Reducing Inter-World Data Copying Overhead and Multi-Threading

Partitioning layers for dual-world processing introduces frequent data transfers between the normal and secure worlds, which can lead to substantial overhead. To address this, SelTZ classifies data generated in each layer as either *sensitive* or *non-sensitive*. Sensitive data, which requires protection, is stored and processed in secure memory, while non-sensitive data resides in a *shared memory region* accessible to both worlds, reducing the volume of data that must be copied to TZ.

While partitioning computation between worlds can increase context switching and world change overhead, SelTZ mitigates this through multi-threaded execution. Instead of frequent world changes for each operation, separate threads in each world process their assigned portions of the computation concurrently. The normal world thread processes non-sensitive data and writes its partial results to the shared memory, while the secure world thread handles sensitive computations and stores its results in the secure memory region. Since these write operations target different memory regions, they can proceed without memory conflicts. The combining of partial results is performed exclusively in the secure world, where it reads from both memory regions and performs in-place replacement in secure memory. The detailed combining process will be elaborated in the following subsections.

To efficiently manage data transfer between worlds, SelTZ implements the shared memory region using a circular buffer structure:

- **Memory Management**: The shared buffer is configured as:

$$\text{Buffer}_{\text{shared}} = \{b_1, b_2, ..., b_N\}, \quad \text{size}(b_i) = B \text{ MB}$$

 doi:10.20944/preprints202411.1852.v1

where N blocks of size B MB are allocated. These parameters (N and B) can be adjusted based on the target device platform's memory capacity. The total shared buffer size ($N \times B$) should be sufficient to accommodate the largest intermediate result size among all layers. For instance, in AlexNet, the largest layer (fifth convolutional layer) produces intermediate results of 384 channels with 13×13 feature maps, requiring approximately 0.25MB for single-precision floating-point storage. Accordingly, on our test platform, we use $N = 2$ blocks of $B = 1$ MB each. When targeting different network architectures, these parameters should be adjusted based on their maximum intermediate layer size - for example, deeper networks with larger feature maps would require proportionally larger buffer allocations. If the intermediate data exceeds the available shared memory space, the computation can be further divided into smaller subsets, processing the data in multiple passes while maintaining the same memory recycling strategy.

- **Thread Synchronization**: When the computational workload is not evenly distributed between worlds (which often occurs due to security requirements), mutex-based synchronization is employed to handle the timing differences in computation completion. This ensures that partial results from the faster thread wait for the slower thread before the combining operation begins in the secure world.

This integrated approach of shared memory and multi-threading minimizes both data transfer and world transition overheads. While TZ handles sensitive operations, the normal world performs non-sensitive computations concurrently, reducing idle time in each environment and optimizing performance. For example, while convolutional operations on sensitive data are executed in the secure world thread, the normal world thread can process non-sensitive portions of the same layer in parallel. This approach also allows larger models to be protected within TZ by limiting the volume of sensitive data managed in secure memory, which is critical given TZ's constrained memory capacity.

3.2.3. Partitioning and Combining Convolutional Layers

Partitioning computations across worlds requires secure combining of intermediate results from each world. In SelTZ, this challenge is handled by selectively partitioning *weights* rather than input data for convolutional layers. Partitioning weights reduces the memory overhead that would arise from tracking large input data in both worlds.

For a convolutional layer with input X, the partitioned weights and biases (W_n, b_n for normal world and W_s, b_s for secure world) are used to compute convolution operations explicitly as:

$$\text{Conv}(X, W_n, b_n)[i,j] = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} X[i+k, j+l] W_n[k,l] + b_n[i]$$

$$\text{Conv}(X, W_s, b_s)[i,j] = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} X[i+k, j+l] W_s[k,l] + b_s[i]$$

where K and L are the kernel dimensions. The zero-padded structure of W_n and W_s allows for efficient SIMD processing. Using ARM NEON instructions, we process multiple elements simultaneously by loading four floating-point values into 128-bit NEON registers. The presence of zeros in the padded weights is particularly advantageous as it allows for sparse computation optimization - when a weight block contains zeros due to partitioning, those multiplications can be skipped entirely, further accelerating the convolution operation. This optimization is implemented using NEON's conditional execution capabilities, effectively reducing the number of required floating-point operations while maintaining the parallel processing advantage.

- **Partitioned Weight and Convolution Calculation with Zero Padding**: The convolution operations are performed in parallel:

$$C_n = \text{Conv}(X, W_n, b_n), \quad C_s = \text{Conv}(X, W_s, b_s)$$

 doi:10.20944/preprints202411.1852.v1

Here, C_n and C_s represent the partial convolution results based on normal-world and secure-world weights and biases, stored in shared and secure memory, respectively.

- **Secure Summation of Partitioned Convolution Outputs**: Results are combined in TZ using element-wise addition:

$$C_{\text{combined}} = C_s + C_n$$

This operation is performed within TZ's secure memory space. Specifically, the secure world process directly accesses C_s from its secure memory allocation and reads C_n from the shared memory region. The element-wise addition is performed in-place where $C_s \leftarrow C_s + C_n$ using NEON SIMD instructions to process multiple elements simultaneously, and this modified C_s becomes C_{combined}, avoiding additional memory allocation. Finally, the shared memory region containing C_n is cleared to prevent potential data leakage. This in-place computation strategy minimizes memory usage within the constrained TZ environment while maintaining security guarantees.

- **Activation and Pooling within Secure Memory**: The activation function f (typically ReLU) is applied to C_{combined} in TZ:

$$A_s = f(C_{\text{combined}}) = \max(0, C_{\text{combined}})$$

For max pooling with window size $k \times k$ and stride s:

$$P_s[i,j] = \max_{p,q \in k \times k} A_s[s \cdot i + p, s \cdot j + q]$$

3.2.4. Partitioning and Combining Fully Connected Layers

Fully connected (FC) layers require a modified approach due to their dense connectivity patterns. The weights and biases are partitioned using a block-wise strategy:

- **Weight Matrix Blocking**: For an FC layer with input dimension d_{in} and output dimension d_{out}, the weight matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$ is divided into blocks:

$$W = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1k} \\ B_{21} & B_{22} & \cdots & B_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mk} \end{bmatrix}$$

where each block B_{ij} is assigned to either W_n or W_s based on its sensitivity score.

- **Partitioned Weight and Bias Computation with Zero Padding**: The computations are performed block-wise:

$$Y_n = \sum_{i,j \in \mathcal{N}} B_{ij} X_j + b_n, \quad Y_s = \sum_{i,j \in \mathcal{S}} B_{ij} X_j + b_s$$

where $\mathcal{N}$ and $\mathcal{S}$ are the sets of block indices assigned to normal and secure worlds respectively, and:

$$b_n[i] = \begin{cases} b[i] & \text{if block row } i \text{ is in } \mathcal{N} \\ 0 & \text{otherwise} \end{cases}$$

$$b_s[i] = \begin{cases} b[i] & \text{if block row } i \text{ is in } \mathcal{S} \\ 0 & \text{otherwise} \end{cases}$$

The block-wise computation structure enables further performance optimization through NEON SIMD instructions, processing multiple elements simultaneously in both secure and normal worlds, similar to the optimization applied in convolutional layers.

 doi:10.20944/preprints202411.1852.v1

- **Secure Summation of Partitioned Results**: Results are combined in TZ using vector addition:

$$Y_{\text{combined}} = Y_s + Y_n$$

- **Activation in Secure Memory**: The activation function (typically ReLU) is applied within TZ:

$$A_{\text{FC}} = \max(0, Y_{\text{combined}})$$

3.2.5. Secure Computation of Normalization Layers

Normalization layers require special handling due to their global dependencies. For Softmax computation:

- **Complete Computation within Secure World**: The entire Softmax operation is performed in TZ:

$$\text{Softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- **Memory Optimization**: To reduce memory usage and ensure numerical stability, computation is performed in-place with only two scalar temporary variables (m and s):

  1. Compute maximum: $m = \max_i(x_i)$ // scalar temporary
  2. Subtract maximum: $x_i \leftarrow x_i - m$
  3. Compute exponentials: $x_i \leftarrow \exp(x_i - m)$
  4. Compute sum: $s = \sum_i \exp(x_i - m)$ // scalar temporary
  5. Normalize: $x_i \leftarrow \exp(x_i - m)/s$

  This implementation requires only two additional scalar values in secure memory, minimizing the memory overhead while maintaining numerical stability.

3.3. Processing Flow Integration

SelTZ processes deep neural networks through a systematic combination of security-driven partitioning and efficient execution strategies. Starting with sensitivity assessment of each layer, computations are selectively distributed across normal and secure worlds while ensuring that activation outputs and final layer outputs, which are critical for preventing MIAs, remain protected in TrustZone.

The processing flow varies by layer type, reflecting their different security requirements and computational characteristics. Convolutional and fully connected layers leverage parallel processing with partitioned parameters, where normal world results are efficiently transferred through circular shared memory buffers while secure world computations remain isolated. These partial results are then combined securely within TrustZone. In contrast, normalization layers, due to their global dependencies, are processed entirely within the secure world using memory-optimized implementations.

A scheduler coordinates this heterogeneous processing by managing:

- Layer dependencies and execution ordering
- Transitions between parallel and secure-only processing
- Allocation and management of shared memory resources

This integrated approach enables SelTZ to maintain its security guarantees while optimizing performance through efficient resource utilization and parallel processing capabilities.

4. Implementation

4.1. Target Neural Network Models

To comprehensively evaluate the effectiveness of SelTZ, we use three neural network architectures that vary in complexity and depth: **AlexNet**, **VGG-7**, and **ResNet-20**. Each architecture presents unique challenges in terms of memory usage, computational demands, and susceptibility to membership

 doi:10.20944/preprints202411.1852.v1

inference attacks (MIAs). While DarkneTZ originally used ResNet-110, we opted for ResNet-20 which extends DarkneTZ's ResNet-18 implementation with TrustZone-aware operations. This choice allows us to better demonstrate the practical applicability of our approach in resource-constrained TrustZone environments. Additionally, since DarkneTZ is open-source, we conducted all experiments on the same platform under identical conditions, ensuring a fair and direct comparison between SelTZ and DarkneTZ beyond the numbers reported in their paper.

For ease of explanation, we present a detailed breakdown of each model's layers in Tables 1–3. AlexNet has five convolutional layers (with kernel sizes 11, 5, 3, 3, and 3) followed by a fully connected and a softmax layer, where the number of neurons for each convolutional layer is 64, 192, 384, 256, and 256, respectively. VGG-7[1] consists of seven convolutional layers with uniform kernel size of 3, where the number of neurons progressively increases (64, 64, 124, 124, 124, 124, 124), followed by a fully connected layer and a softmax layer. ResNet-20 introduces residual blocks that enable significantly deeper architectures. It begins with a 7×7 convolutional layer with 64 filters and stride 2, followed by a 2×2 max pooling layer. The network consists of five stages with increasing channel dimensions (64, 128, 256, and 512), where the transitions between stages use strided convolutions for spatial reduction. Each residual block contains two 3×3 convolutional layers connected by a skip connection, incorporating TrustZone-aware operations throughout the network.

Table 1. Architecture and Partitioning of AlexNet with SelTZ Design

Index	Type	Filter/Channels	Neurons	Partition (World)
1	Convolution	11×11 / 3→64	64	Normal/Secure
2	ReLU	-	64	Secure
3	*Max Pooling*	2×2	64	Secure
4	Convolution	5×5 / 64→192	192	Normal/Secure
5	ReLU	-	192	Secure
6	*Max Pooling*	2×2	192	Secure
7	Convolution	3×3 / 192→384	384	Normal/Secure
8	ReLU	-	384	Secure
9	Convolution	3×3 / 384→256	256	Secure
10	ReLU	-	256	Secure
11	Convolution	3×3 / 256→256	256	Secure
12	ReLU	-	256	Secure
13	*Max Pooling*	2×2	256	Secure
14	FC	-	{100, 200}*	Normal/Secure
15	Softmax	-	{100, 200}*	Secure

*100 classes for CIFAR-100, 200 classes for ImageNet-Tiny

[1] The VGG-7 architecture follows the configuration from DarkneTZ (https://github.com/mofanv/tz_datasets.git)

 doi:10.20944/preprints202411.1852.v1

Table 2. Architecture and Partitioning of VGG-7 with SelTZ Design

Index	Type	Filter/Channels	Neurons	Partition (World)
1	Convolution	3×3 / 3→64	64	Normal/Secure
2	ReLU	-	64	Secure
3	Convolution	3×3 / 64→64	64	Secure
4	ReLU	-	64	Secure
5	*Max Pooling*	2×2	64	Secure
6	Convolution	3×3 / 64→124	124	Normal/Secure
7	ReLU	-	124	Secure
8	Convolution	3×3 / 124→124	124	Secure
9	ReLU	-	124	Secure
10	*Max Pooling*	2×2	124	Secure
11	Convolution	3×3 / 124→124	124	Normal/Secure
12	ReLU	-	124	Secure
13	Convolution	3×3 / 124→124	124	Secure
14	ReLU	-	124	Secure
15	*Max Pooling*	2×2	124	Secure
16	Convolution	3×3 / 124→124	124	Normal/Secure
17	ReLU	-	124	Secure
18	*Dropout*	-	124	Secure
19	FC	-	{100, 200}*	Normal/Secure
20	Softmax	-	{100, 200}*	Secure

*100 classes for CIFAR-100, 200 classes for ImageNet-Tiny

Table 3. Architecture and Partitioning of ResNet-20 with SelTZ Design

Index	Type	Filter/Channels	Neurons	Partition (World)
1	Convolution	7×7 / 3→64	64	Normal/Secure
2	*Max Pooling*	2×2	64	Secure
Stage 1: 64 channels				
3.1	Convolution	3×3 / 64→64	64	Normal/Secure
3.2	ReLU	-	64	Secure
3.3	Convolution	3×3 / 64→64	64	Secure
3.4	Add	-	64	Secure
3.5	ReLU	-	64	Secure
...	(Additional blocks with 64 channels)			
Stage 2: 128 channels				
6.1	Convolution (s=2)	3×3 / 64→128	128	Secure
6.2	ReLU	-	128	Secure
6.3	Convolution	3×3 / 128→128	128	Secure
6.4	Add	-	128	Secure
6.5	ReLU	-	128	Secure
...	(Additional blocks with 128 channels)			
Stage 3: 256 channels				
9.1	Convolution (s=2)	3×3 / 128→256	256	Secure
...	(Similar pattern with 256 channels)			
Stage 4: 512 channels				
12.1	Convolution (s=2)	3×3 / 256→512	512	Secure
...	(Similar pattern with 512 channels)			
15	*Global AvgPool*	-	512	Secure
16	FC	-	{100, 200}*	Normal/Secure
17	Softmax	-	{100, 200}*	Secure

*100 classes for CIFAR-100, 200 classes for ImageNet-Tiny

Our implementation carefully considers the information flow between Normal and Secure Worlds to prevent any potential data leakage. The key insight is that convolutional layers can be safely partitioned between Normal and Secure Worlds when preceded by *guarding layers* such as max pooling that serve as information-reducing operations. In such cases, the Normal World only observes downsampled data for the input, making it impossible to deduce the complete information (i.e., ReLu output). Each table entry specifies the layer type, filter size, output shape, and its partition (world). Following this principle, ReLU and pooling layers after convolution are processed entirely in the Secure World to protect sensitive activation outputs. Similarly, when ReLU outputs directly feed into

 doi:10.20944/preprints202411.1852.v1

another convolutional layer without intermediate guarding operations, the entire operation must be computed in the Secure World.

4.2. Datasets

The experiments use CIFAR-100 and ImageNet Tiny datasets. CIFAR-100 contains 60,000 images across 100 classes, with 500 training images and 100 test images per class. ImageNet Tiny is a scaled-down version of ImageNet, consisting of 100,000 training images and 10,000 validation images across 200 classes, with images resized to 64×64 pixels for edge device deployment.

For membership inference attack (MIA) experiments, we follow DarkneTZ's methodology for dataset construction. For CIFAR-100, we use 25,000 training set samples as member data and 5,000 test set samples as non-member data for attack model training. Evaluation uses 5,000 different samples each from training (members) and test sets (non-members). For ImageNet Tiny, we use 50,000 training samples as member data and 5,000 validation samples as non-member data for training, with 5,000 different samples each from training and validation sets for evaluation.

4.3. Attack Model

We adopt the same membership inference attack model architecture from DarkneTZ but focus solely on activation outputs. The model employs fully connected network (FCN) components with one ReLU-activated hidden layer and 0.2 dropout to process each target model layer's activation outputs. These outputs are concatenated and processed by a final encoder for membership prediction. Training uses Adam optimizer with 0.0001 learning rate for 200 epochs, selecting the model with highest testing accuracy. All experiments use a batch size of 64.

5. Experiment Results

5.1. Experimental Setup

The experiments are conducted on a Raspberry Pi 3B+ platform equipped with ARM Cortex-A53 cores and ARM TrustZone technology. This platform is identical to that used in the DarkneTZ framework, ensuring a fair comparison of TrustZone-related overhead and memory constraints. Being a resource-constrained edge device, the Raspberry Pi 3B+ provides a realistic testbed for evaluating both the security benefits and performance implications of TrustZone deployment in practical scenarios. OP-TEE is used as the TrustZone operating system, enabling the secure execution of selected layers within TrustZone's memory. Models are deployed with layers allocated to either the secure world or the normal world based on SelTZ's selective layer protection strategy.

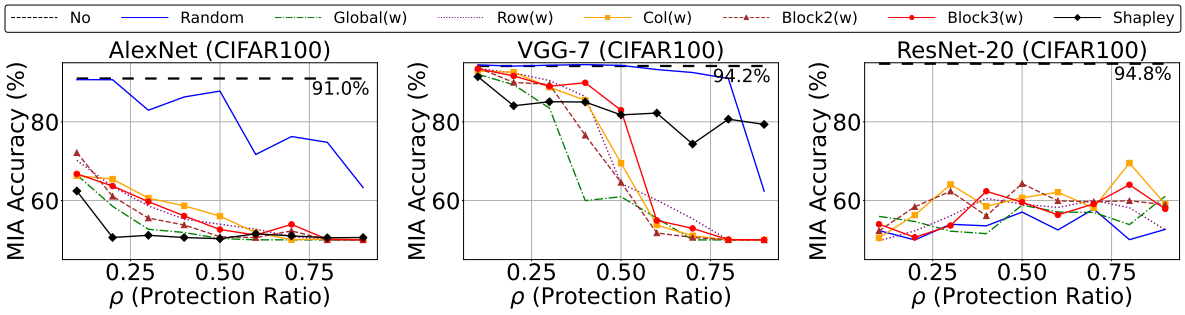
Our implementation partitions layers where necessary, allowing sensitive portions (such as critical activation outputs and specific parameters) to be processed within the Secure World while handling non-sensitive portions in the Normal World to optimize resource usage. This fine-grained approach enables efficient use of TrustZone resources while maintaining robust defenses against MIA risks. For the evaluation of membership inference attacks, we assume an adversary who captures activation outputs observable in the Normal World during inference on the test platform. These captured outputs are then analyzed offline using pre-trained attack models on a separate machine, reflecting a realistic scenario where the adversary collects exposed intermediate outputs from the edge device before performing computationally intensive analysis on more capable hardware.

5.2. Effectiveness of Weight Protection Strategies

We first evaluate different weight protection strategies against membership inference attacks on CIFAR-100. As shown in Figure 2(a-c), the baseline accuracy without protection reaches 91.0%, 94.2%, and 94.8% for AlexNet, VGG-7, and ResNet-20 respectively, indicating significant privacy risks. All protection strategies described in Section 3.2.1 demonstrate substantial improvement over this baseline. While Shapley value-based weight selection provides theoretically optimal importance measurement and shows rapid convergence to random guess (50%) in AlexNet (Figure 2(a)), its computational

 doi:10.20944/preprints202411.1852.v1

overhead becomes prohibitive for more complex architectures. For VGG-7 and ResNet-20, even with 100 Monte Carlo samples, Shapley value computation fails to effectively reduce MIA accuracy regardless of protection ratio (Figure 2(b-c)), requiring significantly more samples and computational time to achieve meaningful results.

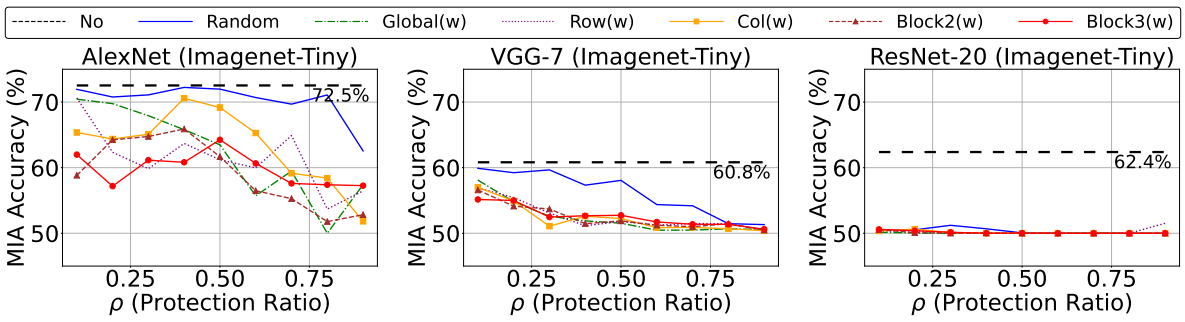


(a) AlexNet (CIFAR-100) **(b)** VGG-7 (CIFAR-100) **(c)** ResNet-20 (CIFAR-100)

Figure 2. Membership inference attack accuracy comparison across different protection ratios (ρ) and partitioning strategies. The baseline (No) shows the vulnerability of unprotected models.

As a practical alternative, our Global(w) selection strategy based on weight absolute values shows consistently strong performance across different architectures, as evident in Figure 2. Among the structured pattern selections, Block2(w) outperforms Block3(w), though neither achieves the stability and effectiveness of global importance-based selection. Interestingly, for ResNet-20 (Figure 2(c)), even random selection achieves near-random guess accuracy (approximately 50%) with just 10% protection ratio, suggesting that its complex architecture with residual connections may distribute sensitive information more evenly across weights, making random protection surprisingly effective.

We further validate our approach on ImageNet-Tiny, with results shown in Figure 3. The baseline vulnerabilities are notably lower (72.5%, 60.8%, and 62.4% for AlexNet, VGG-7, and ResNet-20), suggesting that membership inference attacks face greater challenges with larger, more complex datasets. Notably, pattern-based approaches considering locality show superior performance compared to global selection at low protection ratios, as particularly visible in Figure 3(b-c), though the protection patterns remain generally consistent with CIFAR-100 results. Attack accuracy converges to near-random guess at lower ratios compared to CIFAR-100, suggesting that less secure memory might be needed to achieve adequate protection on ImageNet-Tiny. These results demonstrate our approach's effectiveness across different dataset complexities and validate the robustness of our protection strategies.



(a) AlexNet (ImageNet-Tiny) **(b)** VGG-7 (ImageNet-Tiny) **(c)** ResNet-20 (ImageNet-Tiny)

Figure 3. Membership inference attack accuracy on ImageNet-Tiny dataset shows similar protection patterns but with lower initial vulnerability compared to CIFAR-100.

5.3. Performance Analysis

We evaluate SelTZ's computational efficiency against DarkneTZ across different architectures on our Raspberry Pi 3B+ testbed. As shown in Figure 4, all architectures demonstrate significant

 doi:10.20944/preprints202411.1852.v1

performance improvements. AlexNet execution time reduces from 14.16 seconds with DarkneTZ to 8.86 seconds with SelTZ at $\rho = 0.1$ (1.6× speedup). At $\rho = 0.9$, execution time increases to 12.36 seconds while still maintaining a 1.15× speedup. VGG-7 shows more dramatic improvement, dropping from 155.41ms to 21.41ms at $\rho = 0.1$ (7.3× speedup), with minimal increase to 22.63ms at $\rho = 0.9$ (6.9× speedup). ResNet-20 follows similar trends, improving from 58.36 seconds to 40.70 seconds at $\rho = 0.1$ (1.43× speedup) and scaling to 43.22 seconds at $\rho = 0.9$ (1.35× speedup).

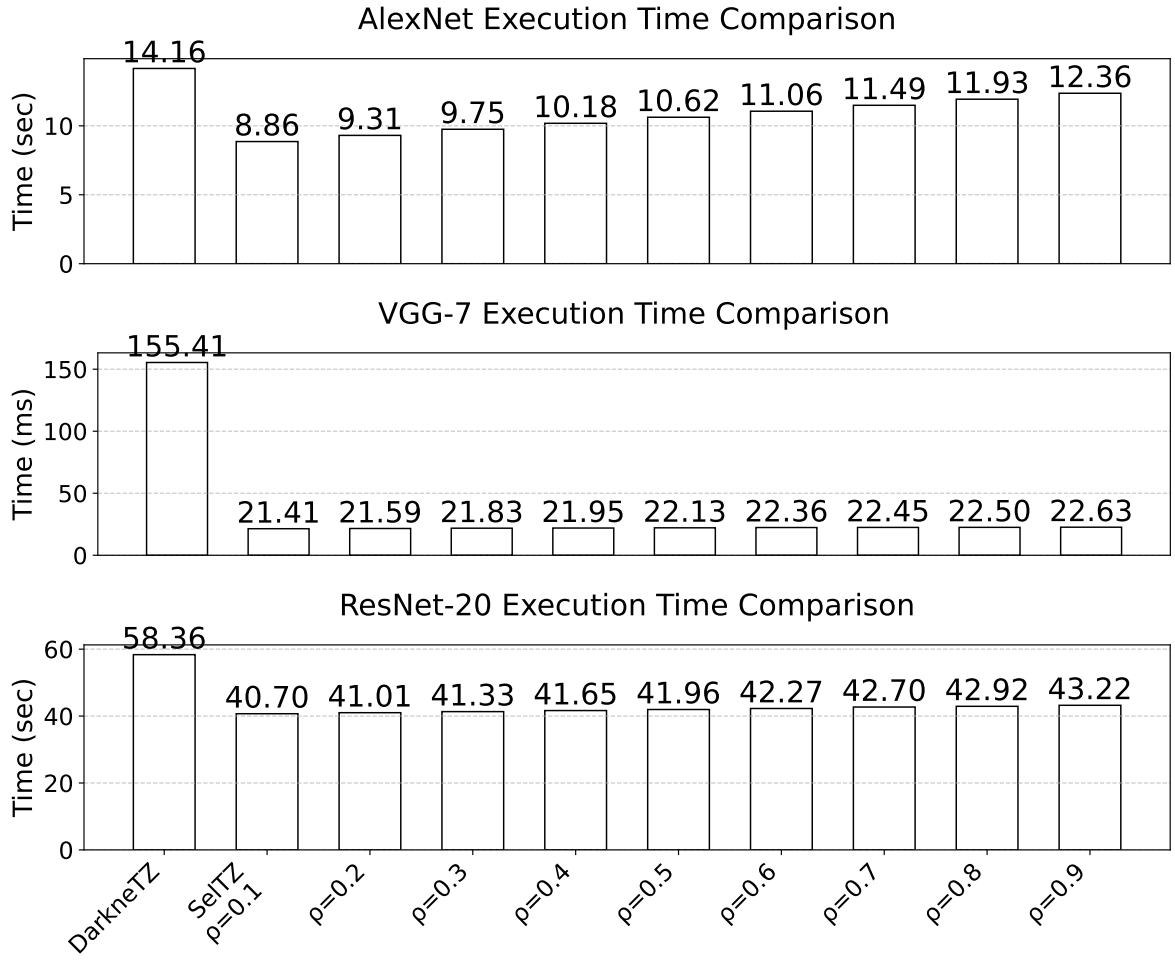


Figure 4. Execution time comparison between DarkneTZ and SelTZ with different protection ratios (ρ).

The variation in speedup across architectures can be attributed to their different baseline implementations in DarkneTZ. Due to Secure World memory constraints, DarkneTZ only protects the last four layers of AlexNet and the last two layers of ResNet-20 in the Secure World, while processing the remaining layers in the faster Normal World. In contrast, VGG-7's relatively smaller size allows DarkneTZ to protect all layers in the Secure World, resulting in a higher baseline execution time and consequently more dramatic improvements with SelTZ. It's worth noting that SelTZ maintains comprehensive protection for all security-critical layers in the Secure World while selectively offloading only those layer computations that can be safely exposed to the Normal World, as detailed in Section 4.1.

These improvements stem from several key optimizations in SelTZ's design. First, selective weight protection significantly reduces TrustZone resource usage compared to DarkneTZ's layer-wise approach. Second, our zero-padding strategy for partitioned weights enables efficient use of ARM NEON SIMD instructions with conditional execution - when a weight block contains zeros due to partitioning, those multiplications can be skipped entirely, accelerating both convolution and fully connected layer operations. Third, our block-wise computation structure further optimizes performance through effective memory locality and reduced world-switching overhead. The relationship between

 doi:10.20944/preprints202411.1852.v1

execution time and protection ratio across architectures enables predictable performance scaling, allowing system designers to make informed tradeoffs between privacy protection and computational efficiency.

5.4. Memory Usage Analysis

TrustZone secure memory usage was measured across all three neural network architectures, comparing DarkneTZ and SelTZ approaches. For fair comparison in the inference scenario, we modified the original DarkneTZ implementation by removing all training-related memory allocations. Even after these optimizations, DarkneTZ's memory overhead stems from its need to pre-allocate secure world memory for all protected layers' data, with each layer requiring sufficient memory for both feature maps and parameters. This approach leads to memory allocation of 1.34 MB for VGG-7, 7.46 MB for AlexNet, and 3.93 MB for ResNet-20. In contrast, SelTZ optimizes memory usage through dynamic allocation based on maximum layer size and in-place operations, requiring only 0.39 MB for VGG-7, 2.80 MB for AlexNet, and 1.96 MB for ResNet-20. This represents substantial memory reductions of 71.13%, 62.41%, and 50.16% respectively.

Table 4. Memory Usage Comparison between DarkneTZ and SelTZ

Model	DarkneTZ (MB)	SelTZ (MB)	Reduction (%)
VGG-7	1.3437	0.3879	-71.13
AlexNet	7.4563	2.8028	-62.41
ResNet-20	3.9338	1.9607	-50.16

The efficiency gains from SelTZ's approach are particularly evident in memory-constrained environments. While DarkneTZ can only protect up to the last four layers in AlexNet and the last two layers in ResNet-20 due to memory limitations, SelTZ's optimized allocation strategy enables protection of significantly more layers. This is achieved by limiting memory demand to the size of the largest layer rather than the cumulative size of all protected layers, while in-place operations further reduce data transfer overhead between Normal and Secure Worlds. These optimizations make SelTZ particularly well-suited for resource-constrained edge devices where memory limitations are a critical concern, while maintaining robust protection against membership inference attacks.

6. Conclusion

In conclusion, this paper presents SelTZ, a novel selective layer protection method that leverages ARM TrustZone to secure deep neural network inference on resource-constrained edge devices. By partitioning layer computations and selectively protecting only the most sensitive data, SelTZ overcomes key limitations of existing solutions that rely on full-layer protection. Our approach achieves significant improvements in both performance and memory efficiency - up to 7.3× speedup and 71% memory reduction compared to DarkneTZ - while maintaining strong privacy guarantees against membership inference attacks. Our experimental results demonstrate that selective protection with global importance-based weight selection provides robust defense against MIAs across different architectures and datasets, reducing attack success rates from over 90% to near random guess (50%). The efficient use of NEON SIMD instructions through zero-padded weight partitioning, combined with parallel processing across security boundaries, enables SelTZ to protect substantially more layers than previous approaches within TrustZone's limited secure memory. Future work could explore several promising directions: (1) adaptive protection ratio adjustment based on runtime privacy requirements and resource availability, (2) extension of our selective protection approach to other types of privacy attacks beyond MIAs, and (3) integration with emerging hardware security features in next-generation edge processors. Additionally, investigating the applicability of SelTZ to more complex model architectures like transformers could further demonstrate its versatility for securing edge AI deployments.

 doi:10.20944/preprints202411.1852.v1

Author Contributions: Conceptualization, methodology, writing-original draft, attack experiments, H.O.; software, investigation, data curation, performance experiments, S.J.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2022-00166529) , by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2024-00337414, Binary Micro-Security Patch Technology Applicable with Limited Reverse Engineering Capability under SW Supply Chain Environments).

Data Availability Statement: The data used for experimental comparisons in this study, referring to the comparison figures, can be found in related research papers. Our implementation code is protected under the proprietary rights of the funding project's institution and therefore cannot be made publicly available.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ARM	Advanced RISC Machines
Conv	Convolutional Layer
DNN	Deep Neural Network
FC	Fully Connected Layer
IoT	Internet of Things
MIA	Membership Inference Attack
NEON	ARM Advanced SIMD Extension
OP-TEE	Open Portable Trusted Execution Environment
ReLU	Rectified Linear Unit
SelTZ	Selective TrustZone Protection
SIMD	Single Instruction, Multiple Data
Softmax	Softmax Function
TEE	Trusted Execution Environment
TZ	TrustZone

References

1. Merenda, M.; Porcaro, C.; Iero, D. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors* **2020**, *20*. doi:10.3390/s20092533.
2. Shokri, R.; Stronati, M.; Song, C.; Shmatikov, V. Membership Inference Attacks Against Machine Learning Models. 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 3–18. doi:10.1109/SP.2017.41.
3. Marcolla, C.; Sucasas, V.; Manzano, M.; Bassoli, R.; Fitzek, F.H.P.; Aaraj, N. Survey on Fully Homomorphic Encryption, Theory, and Applications. *Proceedings of the IEEE* **2022**, *110*, 1572–1609. https://doi.org/10.1109/JPROC.2022.3205665.
4. Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H.B.; Mironov, I.; Talwar, K.; Zhang, L. Deep Learning with Differential Privacy. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; Association for Computing Machinery: New York, NY, USA, 2016; CCS '16, p. 308–318. doi:10.1145/2976749.2978318.
5. Ngabonziza, B.; Martin, D.; Bailey, A.; Cho, H.; Martin, S. TrustZone Explained: Architectural Features and Use Cases. 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), 2016, pp. 445–451. doi:10.1109/CIC.2016.065.
6. Islam, M.S.; Zamani, M.; Kim, C.H.; Khan, L.; Hamlen, K.W. Confidential Execution of Deep Learning Inference at the Untrusted Edge with ARM TrustZone. Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy; Association for Computing Machinery: New York, NY, USA, 2023; CODASPY '23, p. 153–164. doi:10.1145/3577923.3583648.
7. Mo, F.; Shamsabadi, A.S.; Katevas, K.; Demetriou, S.; Leontiadis, I.; Cavallaro, A.; Haddadi, H. DarkneTZ: towards model privacy at the edge using trusted execution environments. Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services; Association for Computing Machinery: New York, NY, USA, 2020; MobiSys '20, p. 161–174. doi:10.1145/3386901.3388946.

8. Nasr, M.; Shokri, R.; Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 739–753. doi:10.1109/SP.2019.00065.
9. Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M.; van der Maaten, L. CrypTen: Secure Multi-Party Computation Meets Machine Learning. Advances in Neural Information Processing Systems; Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; Vaughan, J.W., Eds. Curran Associates, Inc., 2021, Vol. 34, pp. 4961–4973.
10. Liang, J.; Pang, R.; Li, C.; Wang, T. Model Extraction Attacks Revisited. Proceedings of the 19th ACM Asia Conference on Computer and Communications Security; Association for Computing Machinery: New York, NY, USA, 2024; ASIA CCS '24, p. 1231–1245. doi:10.1145/3634737.3657002.
11. Fredrikson, M.; Jha, S.; Ristenpart, T. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security; Association for Computing Machinery: New York, NY, USA, 2015; CCS '15, p. 1322–1333. doi:10.1145/2810103.2813677.
12. Ganju, K.; Wang, Q.; Yang, W.; Gunter, C.A.; Borisov, N. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security; Association for Computing Machinery: New York, NY, USA, 2018; CCS '18, p. 619–633. doi:10.1145/3243734.3243834.
13. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. Proceedings of the 31st International Conference on Neural Information Processing Systems; Curran Associates Inc.: Red Hook, NY, USA, 2017; NIPS'17, p. 4768–4777.