Optimizing Confidential Deep Learning for Real-Time Systems

MOHAMMAD FAKHRUDDIN BABAR, Washington State University, USA
MONOWAR HASAN, Washington State University, USA

Deep neural networks (DNNs) are increasingly used in time-critical, learning-enabled cyber-physical applications such as autonomous driving and robotics. Despite the growing use of various deep learning models, protecting DNN inference from adversarial threats while preserving model privacy and confidentiality remains a key concern for resource and timing-constrained autonomous cyber-physical systems. One potential solution, primarily used in general-purpose systems, is the execution of the DNN workloads within *trusted enclaves* available on current off-the-shelf processors. However, ensuring temporal guarantees when running DNN inference within these enclaves poses significant challenges in real-time applications due to *(a)* the large computational and memory demands of DNN models and *(b)* the overhead introduced by frequent context switches between "normal" and "trusted" execution modes. This paper introduces new time-aware schemes for dynamic (EDF) and fixed-priority (RM) schedulers to preserve the confidentiality of DNN tasks by running them inside trusted enclaves. We first propose a technique that *slices* each DNN layer and runs them sequentially in the enclave. However, due to the extra context switch overheads of individual layer slices, we further introduce a novel *layer fusion* technique. Layer fusion improves real-time guarantees by grouping multiple layers of DNN workload from multiple tasks, thus allowing them to fit and run concurrently within the enclaves while maintaining timing constraints. We implemented and tested our ideas on the Raspberry Pi platform running a DNN-enabled trusted operating system (OP-TEE with DarkNet-TZ) and three DNN architectures (AlexNet-squeezed, Tiny Darknet, YOLOv3-tiny). Compared to the layer-wise partitioning approach, layer fusion can *(a)* schedule up to 3x more tasksets for EDF and 5x for RM and *(b)* reduce context switches by up to 11.12x for EDF and by up to 11.06x for RM.

CCS Concepts: • **Computer systems organization → Real-time systems**.

Additional Key Words and Phrases: DNN, TrustZone

ACM Reference Format:
Mohammad Fakhruddin Babar and Monowar Hasan. 2025. Optimizing Confidential Deep Learning for Real-Time Systems. *ACM Trans. Cyber-Phys. Syst.* 1, 1 (March 2025), 27 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The rapid advancement of IoT applications, including autonomous vehicles, drones, and cognitive robots, alongside improvements in computing power and hardware efficiency, has accelerated the

This research is an extended version of our ECRTS'24 paper [11]. Our preliminary study enables confidential DNN for the EDF scheduler. This paper extends early work to a fixed-priority (RM) scheduler. To incorporate DNN tasks for the RM scheduler and ensure confidentiality, we develop new timing analysis, algorithms, and examples (see Section 4-Section 5). We further redesigned our experiments and performed a comprehensive evaluation for the RM scheduler (see Section 6 and Fig. 7-Fig. 9, Fig. 11, Fig. 12e-12h, Fig. 13d- 13f, Fig. 14b). In addition, we updated Section 8 with the most recent related work and made several editorial changes throughout the paper to improve readability.

Authors' Contact Information: Mohammad Fakhruddin Babar, m.babar@wsu.edu, Washington State University, Pullman, WA, USA; Monowar Hasan, monowar.hasan@wsu.edu, Washington State University, Pullman, WA, USA.

ACM 2378-9638/2025/3-ART
https://doi.org/XXXXXXX.XXXXXXX

integration of machine learning models into cyber-physical systems. Many of these learning-enabled cyber-physical systems are also required to meet stringent real-time constraints. For instance, autonomous vehicles must continuously analyze their surroundings and identify objects through deep neural network (DNN) inference chains. Any delay in this object recognition process could jeopardize decision-making, potentially compromising the safety of the vehicle, passengers, and the surrounding environment. Deploying DNN models on field devices, however, introduces a new set of challenges regarding security and confidentiality. Many autonomous cyber-physical systems frequently handle sensitive data, such as location information, reconnaissance imagery, or medical records. A breach of these systems could lead to significant privacy violations. For instance, a compromised system could expose proprietary models (e.g., parameters, intermediate results, final outputs), thereby leaking the intellectual property of the model provider. Earlier research identified several vulnerabilities, such as membership inference [57, 42], fault injection [20, 41], and input reconstruction [13], that can lead to model compromise and misclassification.

To mitigate these confidentiality risks, researchers have explored executing DNN inference tasks within *trusted enclaves*, such as Intel SGX [4] or ARM TrustZone [7]. However, securely running DNN workloads inside trusted enclaves presents notable difficulties, primarily due to the substantial computing and memory demands of these models, which often exceed enclave memory capacities. For example, a typical image classification task with VGG-16 [59] requires 528 MB of memory, whereas OP-TEE [43], an open-source TrustZone framework for Linux, provides only 16 MB for enclave operations. While efforts have been made to *partition* DNN workloads and execute them within trusted enclaves [10], these methods have largely been designed for general-purpose systems, lacking the necessary real-time guarantees. Adapting existing frameworks without considering periodic, deadline-based real-time tasks will not effectively ensure the dependability requirements of learning-enabled hard real-time systems. For instance, though existing partitioning techniques [53, 62, 46, 24] can reduce memory loads, as discussed in this paper (Section 4.4), frequent switching between trusted and normal execution modes introduces substantial delays due to context-switching overheads, potentially causing critical tasks to miss their deadlines.

In this work, we address the following problem: ***how do we ensure compute-heavy real-time DNN tasks fit in limited capacity enclaves to ensure confidentiality without missing their deadlines?*** To answer this question, we introduce new scheduling models to ensure the confidentiality and temporal constraints of learning-enabled real-time tasks. Our initial approach to making DNN inference tasks both *trusted* and *time-aware* employs a slicing mechanism that partitions DNN models on a *layer-by-layer* basis [62]. This method involves sequentially transmitting one DNN layer at a time to the enclave, executing its computations, and returning the results. However, individual layers may still exceed the enclave's memory capacity due to their size. To mitigate this, we apply Deep Compression [30] to reduce the DNN model size following the enclave's limitations (Section 4.1). We then use the compressed model and enable real-time scheduling capabilities for the existing (non-real-time) layer-wise partitioning idea (Section 4.2 and Section 4.3).

We find that despite real-time guarantees, layer-wise partitioning results in poorer throughput (i.e., fewer tasks are schedulable) due to high context switch overheads (Section 4.4). Hence, we propose a novel "fusion" approach that selectively *groups multiple layers from multiple tasks*, considering enclave capacity and deadline constraints (Section 5). Figure 1 illustrates the key intuition of layer fusion for a three-task system. When DNN layers are sent sequentially to the enclave, extra context switch overheads cause longer response times, and one of the tasks misses

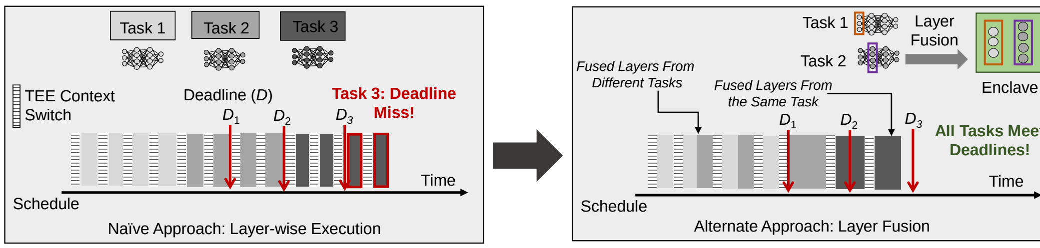


Fig. 1. High-level schematic of the scheduling techniques used in the work. Due to the large size of a DNN model, often it is not feasible to fit within the enclave. Hence, our first approach is to slice the model *layer-by-layer* to fit in the enclave and send them sequentially. However, extra context switch overheads (due to switching back and forth from enclave) may violate real-time constraints. Hence, we also introduce a novel "layer fusion" technique (right rectangle) that *groups multiple layers from multiple tasks* together to reduce context switch costs and results in better schedulability.

the deadline. In contrast, fusing multiple layers saves context switch delays, thus resulting in faster response times. As a result, all tasks meet deadlines.

Our Contributions. This research enables time-aware, confidential DNN execution for learning-enabled real-time systems. Our key contributions include:

- Ensuring *timing guarantees* for performing confidential deep inference in learning-enabled real-time systems.
- Introducing *new scheduling models* for both dynamic (EDF) and fixed-priority (RM) systems to assess the feasibility of deploying real-time DNN tasks on trusted enclaves.

We evaluated proposed techniques on three realistic workloads (e.g., AlexNet-squeezed [32], Tiny Darknet [52], YOLOv3-tiny [5]) running on a Raspberry Pi board [54] and conducted extensive design-space exploration (Section 6). Additionally, we performed a case study using a modified ArduPilot UAV autopilot system [2] with DNN-enabled workloads (YOLOv3-tiny, Tiny Darknet). We found that layer fusion archives better schedulability compared to layer-wise partitioning techniques (Section 6.2).

2 Background

We now start with a background on trusted enclaves (Section 2.1) and DNN architecture (Section 2.2) before introducing our model and related assumptions (Section 3).

2.1 Trusted Execution and ARM TrustZone

Trusted execution environments (TEEs) [55] provide a secure and isolated runtime environment. TEEs guarantee the preservation of confidentiality and integrity for the code and data, preventing any exploitation even in the event of a compromise of the host (i.e., main) operating system. Of the available TEE solutions, Intel SGX [4] and ARM TrustZone [7] are the most widely adopted in industry and research. In this work, we focus on TrustZone due to ARM's dominance in embedded applications.[1]

The runtime operations in TrustZone are divided into "normal" and "secure" worlds, each having its own kernel, user, and memory space (see Fig. 2). In the normal world, a conventional operating system (e.g., Linux/RTOS) provides the execution environment, whereas the secure world uses a

[1]Section 7 further discusses the portability of our approach for SGX.

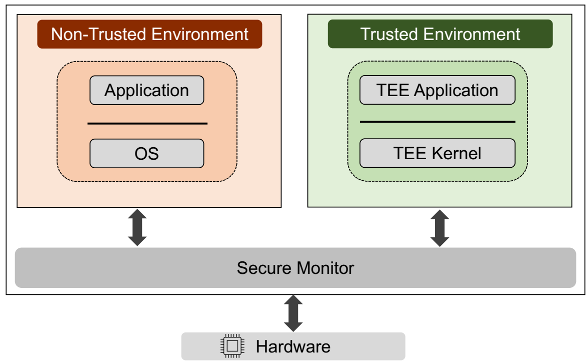


Fig. 2. TrustZone architecture.

minimal trusted kernel (e.g., OP-TEE [43]). The state of the current processor is determined by a specialized bit called the non-secure (NS) bit. The NS bit has two states: NS = 1 for non-secure execution and NS = 0 for secure execution. TrustZone utilizes a mechanism called the secure monitor call (SMC) to transition between these two states. When an SMC instruction is executed in the normal world, the processor cores perform a context switch from the normal world to the secure world, halting operations in the normal world. As a security measure, the normal world is barred from accessing secure memory, while the secure world has access to normal world memory. TrustZone also isolates external peripherals. Prior research provides an extensive survey on TrustZone technology and its applications [51].

2.2 Confidential Deep Neural Inference

DNNs consist of an input layer, one or more hidden layers, and an output layer [58, 47]. Each layer is made up of interconnected nodes, with edges representing the connections between them, each with its own weight and threshold value. Mathematically, a DNN can be represented as a function that maps an input vector $\mathbf{X}$ to an output vector $\mathbf{Y} = \hat{F}(\mathbf{X})$, where $\hat{F}$ is the DNN function. Each layer consists of a set of neurons. Each neuron takes a weighted sum of its inputs and applies an activation function to produce its output. When a node's output exceeds a certain threshold, it is activated, and the data is propagated to the next layer. DNN algorithms build models using training data, allowing them to make predictions without explicit programming. During the *inference* process, input data passes through the layers, each performing matrix multiplications on the data. The final layer can produce numerical or classified outputs, depending on the application. In DNN inference, there is no cross-dependency between any two layers, and each layer can be computed sequentially [34]. We refer the readers to Goodfellow et al. [29] for additional background on DNNs.

Many DNN-based applications (such as image processing, object detection, medical records, and financial transactions) handle sensitive data and must be protected from tampering or theft of intellectual property [25, 56]. To protect model parameters (and hence, ensure "confidentiality"), one emerging approach is to run critical DNN layers inside trusted enclaves such as TrustZone. As enclaves have limited memory and DNN models are typically large [28, 59, 26], one common approach (used in general-purpose systems) is to run the DNN workload layer by layer [62, 48]. This is known as "layer-based partitioning," in which each layer forms an independent partition. Each partition contains weights and biases, which are stored in a separate encrypted file. The decryption key is stored in the enclave. On the secure side, a trusted application decrypts the (encrypted) partition file after loading it into shared memory. In our prior work [10], we surveyed state-of-the-art techniques for enabling confidential deep learning. While researchers explore confidential deep neural inference for general-purpose and mobile/embedded computing systems, surprisingly, there has not been any prior work (except ours [9, 11]) that considers timing constraints and periodic workloads used in real-time applications.

3 Model and Assumptions

3.1 System Model

We consider a uniprocessor real-time system running on a TEE-enabled platform. The system consists of n real-time tasks $\Gamma = \{\tau_1, \ldots \tau_n\}$ performing DNN inference. Each task τ_i is characterized by $\tau_i = \{C_i^a, T_i, D_i, L_i, \mathcal{W}_i\}$, where C_i^a is the worst-case execution time (WCET) of the task inside the enclave, T_i is the period of task τ_i, D_i is the deadline, L_i is the number of layers of the DNN task, and $\mathcal{W}_i$ is the set of sizes of each layer of the DNN task τ_i where $\mathcal{W}_i = \{w_{i1}, w_{i2}, \cdots w_{iL}\}$. Here, w_{ik} is the size of the weights associated with the edges between nodes (neurons), activation, and bias of nodes. In addition, W_i is the size of the DNN task τ_i where $W_i = \sum_{k=1}^{L_i} w_{ik}$. As mentioned earlier (Section 2.2), each layer partition, which includes weights and biases, is stored in an encrypted file. This encrypted file is loaded into shared memory and decrypted by a trusted application on the secure side. Let us denote $C_i^a = C_i^{dec} + C_i^{com}$ as the computation inside the enclave, where C_i^{dec} is the time required for the decryption of the layers information and C_i^{com} is the computation time of task τ_i.

We assume the tasks follow either the earliest deadline first (EDF) [61] or rate-monotonic (RM) [40] scheduling policy. We use EDF and RM scheduling policies as they are the typical schedulers implemented in real-time operating systems and widely used by the real-time research community [15, 14, 16]. EDF is an optimal dynamic priority scheduling algorithm for uniprocessor systems, meaning that if a task set is schedulable under any algorithm, it is also schedulable under EDF [17, 23]. Likewise, RM is an optimal fixed-priority scheduling policy for implicit deadline systems [44] and also the default scheduler for many real-time operating systems such as FreeRTOS [3] and NuttX [1] due to its simplicity. Considering that this research is the first effort to enable confidential DNN for real-time applications, we resort to the most widely used real-time scheduling policies to ensure broader compatibility.

We consider an implicit deadline system (i.e., each task's period is equal to its deadline, $D_i = T_i$). The taskset Γ is "schedulable" if the response time of each task (the time between arrival to completion) is less than its deadline. The trusted enclave has a finite capacity δ, i.e., it can execute $L_i \geq 1$ layers together as long as the total resource requirements of those layers are less than δ. We consider the size of each layer of a task less than δ. Invoking a TEE session involves a series of API calls. For instance, OP-TEE requires 5 API calls for instantiating and terminating a TEE session (see Table 1). Each time the DNN layers enter the enclave, the data needs to be transferred into the enclave. Let $C_{s_i}^{st}$ be the SMC setup time and $C_{s_i}^{d}$ be the SMC cleanup time. Hence, $C_i^{cs} = C_{s_i}^{st} + C_{s_i}^{d}$ captures this data copy overhead. Note that the parameter C_i^{cs} is not part of the worst-case execution time (C_i^a). If a task requires n_i^{cs} many context switches (to-and-from normal to secure world), the total data copy overhead will be $n_i^{cs} \times C_i^{cs}$. In Section 5.2, we derive bounds on the number of context switches.

Existing TEE implementations (for instance, OP-TEE) use non-preemptive enclave execution. We incorporate this behavior, i.e., when a task performs DNN inference inside the enclave, other higher-priority tasks will be "blocked" until the currently running task releases the enclave.

3.2 Adversary Model

The pre-trained model (e.g., parameters, hyperparameters, and architecture of the DNN) is deployed to the real-time platform prior to system operation. We assume an adversary attempting to access sensitive model information. Our focus is on protecting the DNN's inference operations from such threats. While the attacker may access input data, they are unable to retrieve details about the model's architecture or final inference, as these are executed within the enclave. Though attackers may be aware of task periods and execution times, we assume they cannot bypass the TEE's security measures. These assumptions are consistent with prior work [48, 63].

Following the convention, we assume that the pre-trained model's parameters are stored in encrypted form in local storage. The model's hyperparameters, which are typically publicly available and do not expose sensitive information about the input or training data [36], remain unencrypted in the normal world. During inference, when a job is executed, both the input data and the encrypted model parameters are loaded into the enclave memory. The model parameters are then decrypted within the enclave to perform the necessary inference operation.

4 Time-Aware Confidential Deep Learning

In the vanilla case (i.e., when model confidentiality is not a concern), the weights and biases of each neuron in a DNN architecture can be loaded into memory to calculate neuron activation. However, a system with confidentiality requirements (execute models within an enclave) presents challenges when it comes to preloading all the necessary values (e.g., weights, biases) due to limited enclave size, which could be as low as 8 MB for some systems [49]. In contrast, most DNN models need 100+ MB [59]. If a DNN model is too large, then the model may fail to execute inside the enclave. To test this, we conducted a simple experiment on Raspberry Pi running OP-TEE and Darknet [53] that tried to load an AlexNet architecture [37]. For large models (e.g., vanilla AlexNet), Darknet could not load to the model. Hence, we used a model compression technique using *Deep Compression* [30] to reduce the model size (presented next). We repeated the same test with a compressed AlexNet-squeezed model [32] and were able to load and run the model successfully.

4.1 Resizing (Trimming) the Model

Deep Compression is a three-stage pipeline that reduces the storage requirement of neural networks by 35x to 49x without compromising their accuracy. The pipeline consists of pruning, trained quantization, and Huffman coding [49]. The first stage prunes the network by learning only the essential connections, and the second stage quantizes the weights to enforce weight sharing. Finally, the pipeline applies Huffman coding. The method reduces the storage required by AlexNet-squeezed by 35x (from 240 MB to 6.9 MB), and VGG-16 by 49x (from 552 MB to 11.3 MB), without any significant loss of accuracy. This enables the large model to fit inside TEE, tackling the memory constraints.

Recall that, to fit the model in the TEE, the size of each layer must be less than the enclave capacity δ. For a given DNN task τ_i, W_i is the size of the task, L_i is the total number of layers, and then the set of size of the layers is $\mathcal{W}_i = \{w_{i1}, \cdots, w_{iL_i}\}$, where $W_i = \sum_{j=1}^{L_i} w_{ij}$. We check $\forall w_{ij}, w_{ij} < \delta$. If $w_{ij} > \delta$, we calculate the approximation $\theta_{ij} = w_{ij} - \delta$ required for this layer. The approximate percentage is defined by $\theta_{ij}\% = \theta_{ij}/w_{ij}$. The first stage of Deep Compression (see Algorithm 1) prunes the network by learning only the required connections, and the second stage quantizes the weights to enforce weight sharing. In general, for a network with m connections, each connection is represented by b bits, constraining the connections to have only k shared weights

Algorithm 1 Model Compression

```
1: Input: w_ij , λ
2: Output: Compressed Size (w'_ij)
3: Prune the network below a certain threshold λ following state-of-the-art techniques [31].
4: Retrain the network.
5: Quantize the weights of model: r = mb / (m log_2 k + kb)    ▷ Plugging the value of r from approximation percentage (i.e., (1 − θ_ij%) ) to get the value of k
6: Huffman coding to the quantized weights    ▷ final compressed weight
7: return w'_ij
```

Algorithm 2 Resize all Layers

```
1: Input: Model size set (𝒲_i), TEE Capacity δ
2: Output: Resized model size set (𝒲'_i)
3: 𝒲'_i = [ ]    ▷ Initialize to an n empty array
4: for j = 1 to L_i in 𝒲_i do
5:     if w_ij > δ then
6:         Optimized the layer using Algorithm 1
7:         𝒲_i' ← w_ij'
8:     else
9:         𝒲_i' ← w_ij
10:    end if
11:    j ← j + 1
12: end for
13: return Resized model (𝒲'_i)
```

will result in a compression rate of

$$r = \frac{mb}{m \log_2 k + kb}. \tag{1}$$

Let us assume $(1 - \theta_{ij}\%)$ is the desired value for the compression rate r. Plugging the desired compression rate $r = (1 - \theta_{ij}\%)$, we can find the cluster size k based on Eq. (1). After checking and resizing all the layers, we will get the desired task ready that can fit within the enclave (see Algorithm 2).

4.1.1 Formal Description of Model Trimming. Algorithm 1 and Algorithm 2 formally present our ideas for trimming a given DNN model. The model compression process (Algorithm 1) initially prunes the network below a threshold λ to remove less critical connections (Line 3). For this, we use the techniques Han et al. [31] described. We rerun the network to learn the final weights with pruned networks (Line 4). Then, the algorithm quantizes weights, determining the value of shared weights k plugging desired compression rate $r = (1 - \theta_{ij}\%)$ in Eq. (1) (Line 5). Finally, we apply Huffman coding [49] to the quantized weights (Line 6).

Following the steps in Algorithm 1 allows us to resize a single layer. We then use Algorithm 2 to resize *all* the layers of a task τ_i so that we can fit at least a single layer at a given time inside TEE. Algorithm 2 examines each layer of τ_i to determine if it exceeds the TEE capacity δ (Lines 4-12). For instance, if $w_{ij} > \delta$, the layer is optimized using Algorithm 1 (Line 6) and stores the resized layer's information in $\mathcal{W}'_i$ (Line 7). If $w_{ij} < \delta$, unchanged value of w_{ij} is stored in $\mathcal{W}'_i$ (Line 9). This process is repeated for each layer of τ_i, and resized layer information is stored in $\mathcal{W}'_i$.

We note that a compressed model may not fit into TEE due to limited enclave size (i.e., $W_i = \sum w_{ij} > \delta$). In such cases, a known technique (used in general-purpose systems) is to split the DNN model into smaller parts [62, 34]. This partitioning method is beneficial as the only values needed at a given time are the activation of the previous layer, the weights, and biases for the current layer. To illustrate, for two fully connected layers, each with z neurons, it would require z activations, $z \times z$ weights, and z biases. This effectively reduces the instantaneous memory requirement to that of a single layer. The largest layer in the model determines the minimum amount of secure world memory needed for confidential DNN execution. However, this approach partitions each layer and transfers results back and forth from secure to the normal world. This extra context switch overhead could be a bottleneck for real-time applications. Thus, we need timing analysis and schedulability conditions to ensure all tasks retain real-time constraints, as we present below.

4.2 Layer-wise Partitioning for EDF Scheduler (LW-EDF)

We refer to our layer-wise partitioning technique for the EDF scheduler as LW-EDF. Traditional EDF schedulability conditions often involve checking many relative deadline points to assess the schedulability of a taskset up to the hyperperiod [64, 45]. To speed up this process, Zhang et al. propose an improved algorithm (called QPA) that significantly reduces the computation required to check each relative deadline [64]. To determine the schedulability conditions for LW-EDF, we use the existing QPA-based EDF timing analysis technique [64] and adapt it to our DNN-based workload. We choose QPA-based analysis instead of others [45] because *(a)* it provides us a modular model that can be extended to more general tasksets (arbitrary deadline systems) and *(b)* computational complexity of QPA is an offline (design-time) analysis which will not affect runtime performance.

Recall that the execution within the enclave is non-preemptive. Such behavior is modeled by incorporating a "blocking" delay in schedulability analysis. In EDF scheduling with blocking, a set of tasks is schedulable if $\forall t > 0$, $h(t) + b(t) \leq t$, where $h(t)$ is the *processor demand function* and $b(t)$ is the blocking delay [60, 22]. The function $h(t)$ calculates the maximum execution time required by the system for all tasks with both their arrival times and their deadlines in a contiguous interval of length t. The demand function $h(t)$ is given by: $h(t) = \sum_{i=1}^{i=n} \left\lfloor \frac{t}{T_i} \right\rfloor C_i$. In our context, the blocking delay is $b(t) = \max\{C_j^{cs} | D_j > t\}$.

For LW-EDF, the computing time is given by $C_i = C_i^a + n_i^{cs} \times C_i^{cs}$, where n_i^{cs} is the total SMC context switches. Hence, we can rewrite $h(t)$ as follows: $h(t) = \sum_{i=1}^{i=n} \left\lfloor \frac{t}{T_i} \right\rfloor (C_i^a + n_i^{cs} \times C_i^{cs})$, see Lemma 4.1 for a formal derivation. Note that, in LW-EDF, $n_i^{cs} = L_i$. The upper limit of t that needs to be checked is defined by $S = \max\{T_1, T_2, \cdots, T_n\}$. The taskset is schedulable if $U < 1$ and $h(t) + b(t) \leq T_{min}$, where $T_{min} = \min\{T_1, T_2, \cdots, T_n\}$.

Algorithm 3 presents steps for the schedulability checks following EDF scheduling. We start by finding the maximum task period in the taskset (Line 3). T_{min} stores the minimum value of the task period in the taskset (Line 4). The processor demand function $h(t)$ calculates the maximum execution time required by the system for given t (Line 6). If $h(t) + b(t) > T_{min}$ and $h(t) + b(t) < t$, we tighten the bound on processor demand to check if we can execute all ready tasks. This is done by changing the value of t to $h(t)$ (Line 8). If $h(t) + b(t) \leq T_{min}$ at any time t, we can conclude that our system can execute all ready tasks without missing any deadlines. Therefore, the task set is schedulable (Line 9). If it finds $h(t) + b(t) > t$ at any time t, then we report that the taskset is not schedulable (Line 13).

The following lemma shows the expression for processor demand function, $h(t)$.

Lemma 4.1. *The maximum execution time required by the system contiguous interval of length t following EDF scheduling, is given by: $h(t) = \sum_{i=1}^{i=n} \left\lfloor \frac{t}{T_i} \right\rfloor C_i$.*

Proof. From traditional EDF timing analysis [64], $h(t) = \sum_{i=1}^{i=n} \max\{0, 1 + \lfloor \frac{t - D_i}{T_i} \rfloor\} \times C_i$. Replacing $D_i = T_i$ in the above equation (since we have an implicit deadline system) and after simplification,

Algorithm 3 LW-EDF Schedulability

Input: Real-time taskset (Γ)
Output: Taskset schedulability
$t \leftarrow \max\{T_1, T_2, \cdots T_n\}$
$T_{min} \leftarrow \min\{T_1, T_2, \cdots T_n\}$
while $t > T_{\min}$ **do**
  $h(t) \leftarrow \sum_{i=1}^{n} \lfloor \frac{t}{T_i} \rfloor (C_i^a + L_i \times C^{cs})$ ▷ *Calculate $h(t)$ for the given t*
  if $h(t) + b(t) > T_{\min} \wedge h(t) + b(t) < t$ **then**
    $t \leftarrow h(t) + b(t)$
  else if $h(t) + b(t) \leq T_{\min}$ **then**
    Taskset is schedulable
    Break
  else
    Taskset is not schedulable
    Break
  end if
end while

Algorithm 4 LW-RM Schedulability

Input: Real-time taskset (Γ)
Output: Taskset schedulability
$R_i(0) = (C_i^a + n_i^{cs} \times C_i^{cs})$
while $R_i(k+1)! = R_i(k), \forall \tau_i$ (from high to low priority order) **do**
  $R_i(k+1) = b_i + (C_i^a + n_i^{cs} \times C_i^{cs}) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(k)}{T_J} \right\rceil (C_j^a + n_j^{cs} \times C_j^{cs})$
end while
if $R_i \leq D_i$ **then**
  Taskset is schedulable
else
  Taskset is not schedulable
end if

$h(t)$ can be rewritten as: $h(t) = \sum_{i=1}^{i=n} \max\{0, \lfloor \frac{t}{T_i} \rfloor\} \times C_i$. Note that, $\frac{t}{T_i}$ is a non-negative value. Hence, reduced form of $h(t)$ is $h(t) = \sum_{i=1}^{i=n} \lfloor \frac{t}{T_i} \rfloor C_i$. Replacing $C_i = C_i^a + n_i^{cs} \times C_i^{cs}$, $h(t)$ can be rewritten as: $h(t) = \sum_{i=1}^{i=n} \lfloor \frac{t}{T_i} \rfloor \times (C_i^a + n_i^{cs} \times C_i^{cs})$. □

4.3 Layer-wise Partitioning for RM Scheduler (LW-RM)

We refer to the RM variant of the layer-wise partition approach as LW-RM. For RM scheduling, the response time R_i of a task τ_i is calculated iteratively using the following equation: $R_i(k+1) = b_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j(k)}{T_J} \right\rceil C_j$, where $hp(i)$ is the set of task with a priority higher than τ_i and b_i is the blocking delay [22]. The computing time is given by $C_i = C_i^a + n_i^{cs} \times C_i^{cs}$, where n_i^{cs} is the total SMC context switches. Hence, we can rewrite the recurrence relation as follows:

$$R_i(k+1) = b_i + (C_i^a + n_i^{cs} \times C_i^{cs}) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(k)}{T_J} \right\rceil (C_j^a + n_j^{cs} \times C_j^{cs}) \qquad (2)$$

The blocking delay is given by $b_i = \max_{j \in lp(i)} \{C_j\}$, where $lp(i)$ denotes the set of tasks with lower priority that τ_i. The taskset is schedulable if $R_i \leq D_i$.

Algorithm 4 presents steps for the schedulability checks following RM scheduling. We follow busy-window-based analysis [8] for schedulability checking. The algorithm begins by calculating

Table 1. APIs Required for Invoking a TEE Call. The Overheads are Measured on Raspberry Pi 3 Model B.

API	**Function**	**Overhead (μs)**
`TEEC_InitializeContext()`	Initialize connection	240
`TEEC_OpenSession()`	Open a new TEE session	18000
`TEEC_InvokeCommand()`	Invokes a Command	280
`TEEC_CloseSession()`	Close the session	1180
`TEEC_FinalizeContext()`	Close connection	110

the initial response time $R_i(0)$ for a task τ_i (Line 4). Then, the algorithm enters a loop where it iteratively calculates the response time following busy-window based analysis (Line 6). This loop continues until the response time stabilizes (i.e., $R_i(k+1) = R_i(k)$). The algorithm checks for schedulability conditions once the response time is calculated (Line 7-Line 11).

4.4 The Need for Further Optimization

For a given task τ_i, the worst-case execution time of the model inside TEE is C_i^a, where $C_i^a = \sum_{j=1}^{j=L_i} C_{ij}^a$ and C_{ij}^a is the computation time for layer j. In LW-EDF/LW-RM, if a task τ_i has L_i number of layers, we need L_i number of context switches. The total execution time of task τ_i required in the layer-wise approach is $C_i = C_i^a + L_i \times C_i^{cs}$. We now explain the overhead of layer-wise partitioning using a simple example.

Example 1. Let us assume we have three tasks τ_1, τ_2, τ_3 each having 5 layers (i.e., $L_i = 5$) and $\delta = 7$. The size of τ_1 and τ_2 is 10, and the size of τ_3 is 5 units. We cannot execute all the layers of τ_1 inside the enclave as the size of $\tau_1 > \delta$. LW-EDF/LW-RM requires five SMC switches from the normal to secure world for five layers for each task τ_1, τ_2, and τ_3. Hence, we need $3 \times 5 = 15$ SMC switches to execute these three tasks inside TEE.

Despite LW-EDF and LW-RM ensure real-time guarantees (for schedulable tasksets), as we shall see below (and also from our evaluation in Section 6), they result in poorer schedulability. This is because each switching results in extra SMC invocation, which increases task response times. For example, OP-TEE performs five API calls to initiate and teardown a TEE session (See Table 1). Each of these API calls takes a considerable amount of time. We carried out experiments to time each call on a Raspberry Pi platform. As the Table shows, initiating a TEE session, transferring data to/from the enclave, and cleanup steps take approximately 20 ms. In our context, each of the layer execution sessions will add up those TEE API overheads, thus potentially slowing down the inference task and may result in missed deadlines. We further illustrate this issue using an example.

Example 2. Let us consider the taskset listed in Table 2.

Table 2. Example Taskset 1.

Task	L	C^{cs}**/layer**	C_i^a	C	T
τ_1	8	20	290	450	700
τ_2	6	20	270	390	1500
τ_3	8	20	290	450	3000

We now show how layer-wise execution in LW-EDF/LW-RM adds context switch overheads that can increase the overall execution time. There are three tasks τ_1, τ_2, τ_3, where C_1^a, C_2^a, C_3^a are 450, 390, and 450 units respectively. The maximum blocking delay for task τ_2 is 450 time units. The periods

T_1, T_2, T_3 are 700, 1500, and 3000 time units, respectively. In this taskset, $\sum C_i^a/T_i = 0.69 < 1$. Let us assume the context switch overhead is 20 units per layer. Adding this context switch overhead leads execution times, C_1, C_2, C_3 to 450, 390, and 450 units, respectively. As a result, the utilization is $\sum C_i/T_i = 1.05 > 1$. The taskset is not schedulable under LW-EDF/LW-RM since the system utilization is over 100%. In this example, we can see the summation of the actual execution time, $\sum C_i^a = 850$, and the summation of total execution time $\sum C = 1290$. This indicates an additional 34% context switching overhead in executing the taskset.

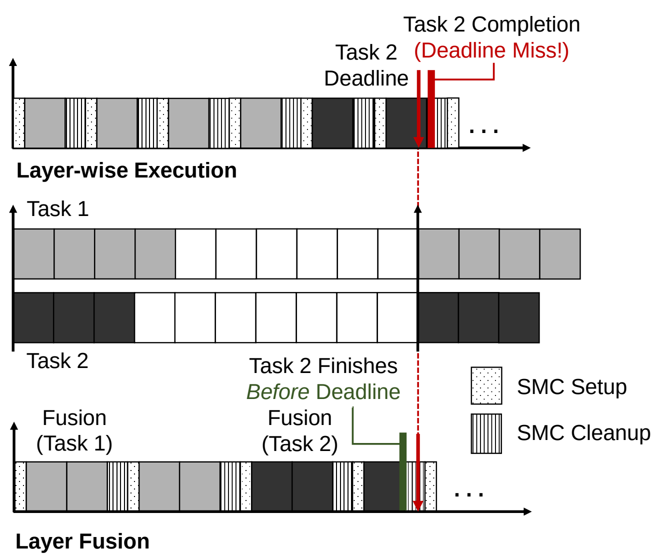


Fig. 3. Key intuition of model fusion: when the tasks are executed layer-wise (top schedule), Task 2 misses the deadline due to multiple context switch overheads. However, in a multi-layer execution approach (bottom schedule), multiple layers are fused, which reduces context switch overheads, and all the tasks meet their deadlines.

To address this problem, we develop a simple yet compelling idea. Instead of sending each layer sequentially, we propose to *group (fuse) multiple layers from multiple tasks (as long as they fit in the enclave) and send them together*. Figure 3 illustrates a high-level schematic for two tasks. In this case, layer-wise execution misses deadlines for Task 2 due to multiple context switch overheads. However, when we fuse the layers in Fusion, we save context switch costs, thus allowing both tasks to meet deadlines.

Task fusion has been used for TEE-enabled conventional (i.e., non-learning enabled) real-time systems to reduce TEE context switch overheads [50]. We borrow a similar concept to group multiple layers of tasks and fit them within the enclave. For each decision instance, we group the tasks based on priorities with the following two goals: *(a)* maximize enclave utilization (capacity usage), i.e., fit as many layers as possible, and *(b)* satisfy timing requirements (deadlines). Our selection process, as described in Section 5, is inspired by the bin-packing heuristics (such as best-fit) [19] used in partitioned multiprocessor scheduling.

5 Multi-layer Task Fusion

Fusing multiple layers from multiple tasks can save context switch overheads compared to the layer-wise partitioning approach. For example, AlexNet-squeezed [28] has 16 layers. If the system follows layer-wise transfer to the enclave, it needs 16 context switches (one for each layer). In contrast, assuming each layer is 1 MB in size and the enclave has 8 MB of memory, if we allow the grouping of layers, we can finish the execution with two context switches. We now illustrate how Fusion improves schedulability. Fusion works independent of task period types (i.e., harmonic/non-harmonic), as illustrated in the following example.

Example 3. Let us consider the following taskset parameters (Table 3 and Table 4).

Table 3. Example Taskset and Layer Size.

Task	L	Size of layers (MB)	Total Size (MB)
τ_1	8	{0.046, 0.186, 0.48, 0.39, 0.27, 5.84, 2.69, 1.50}	11.40
τ_2	6	{0.186, 0.48, 0.39, 5.84, 2.69, 1.50}	11.08
τ_3	8	{0.046, 0.186, 0.48, 0.39, 0.27, 5.84, 2.69, 1.50}	11.40

Table 4. Example Taskset with Fusion Parameters.

Task	L	C^{cs}/layer	CS (fusion)	C_i^a	C	T (non-harmonic)	T (harmonic)
τ_1	8	20	2	290	330	700	700
τ_2	8	20	2	270	310	1500	1400
τ_3	8	20	2	290	330	3000	2800

Case 1: EDF Scheduling. We show for both harmonic and non-harmonic cases. Let us first consider the non-harmonic periods. In this case, $\sum C_i/T_i = 0.78 < 1$. We can calculate the schedulability conditions of as follows: *(a)* $t = 3000$, $h(t) = 2270$; *(b)* $t = 2270, h(t) = 1300$; and *(c)* $t = 1300$, $h(t) = 330$. We can see $h(t) < T_{min}$. Hence, the taskset is schedulable (recall: the same taskset is not schedulable using LW-EDF). For taskset with harmonic periods, *(a)* $t = 2800$, $h(t) = 2270$; *(b)* $t = 2270$, $h(t) = 1300$; and *(c)* $t = 1390$, $h(t) = 330$. We can see $h(t) < T_{min}$. Hence, the taskset is schedulable.

Case 2: RM Scheduling. For the non-harmonic periods, $\sum C_i/T_i = 0.78$. Let us calculate the response time of τ_1 as follows: $R_1(0) = 640$ (here $b_1 = 330$) and $R_1(1) = 640$. Hence, $R_1 = 640$. As $R_1 < D_1 = 700$, τ_1 meets its deadline. For τ_2, $b_2 = 330$ and $R_2(0) = 660$, $R_2(1) = 970$, $R_2(3) = 1280$, and $R_2(4) = 1280$. Hence, $R_2 = 640$ and $R_2 < D_2 = 1500$ (i.e., τ_1 also meets its deadline). Finally, for τ_3, $R_3(0) = 330$ (in this case $b_3 = 0$), $R_3(1) = 970$, $R_3(3) = 1280$, and $R_3(4) = 1280$. Hence, $R_3 = 1280$ and as $R_3 < D_3 = 3000$, τ_3 meets its deadline. Since $R_i < D_i$ for $\forall \tau_i$, the taskset is schedulable (recall: the same taskset is not schedulable using LW-RM).

Let us now consider the harmonic case. For τ_1, $R_1(0) = 640$ and *(b)* $R_1(1) = 640$. Hence, $R_1 = 640$ and $R_1 < D_1 = 700$. For τ_2, $R_2(0) = 660$, $R_2(1) = 970$, $R_2(3) = 1280$; and $R_2(4) = 1280$. Hence, $R_2 = 640$ and $R_2 < D_2 = 1400$. For τ_3, $R_3(0) = 330$, $R_3(1) = 970$, $R_3(3) = 1280$, and $R_3(4) = 1280$. Hence, $R_3 = 1280$ and $R_3 < D_3 = 2800$. Since for each task τ_i, we find $R_i < D_i$, the taskset is schedulable.

5.1 Layer Fusion: Workflow

We call the EDF and RM variants for layer fusion as Fusion-EDF and Fusion-RM, respectively. Our proposed fusion approach aims to maximize the usage of TEE capacity. Hence, we send the

Algorithm 5 Task Fusion and Scheduling

1: **Input:** Real-time taskset (Γ), TEE-capacity δ
2: **Output:**Taskset schedulability decision
3: Compress the Model ▹ *See Algorithm 1*
4: Resize Layers ▹ *See Algorithm 2*
5: $\Omega(t) = \{\mathcal{W}_1', \mathcal{W}_2', \cdots \mathcal{W}_i'\}$ ▹ *Obtain the set of the weight of each task available at time t*
6: T_{hyp} =LCM of $\{T_1, T_2, \cdots, T_n\}$ ▹ *T is the set of period of all DNN tasks*
7: **BEGIN** ▹ *Find layers to send to TEE*
8: **while** TRUE **do**
9:     S=FIND_LAYERS_TO_SEND$\{\Omega(t)\}$ ▹ *See Line 23 for definition*
10:     Send S to TEE
11: **end while**
12: **END**
13: **function** FIND_TRANSITION_OF_LAYERS($\Omega(t)$)
14:     $i \leftarrow$ index of first task in $\Omega(t)$
15:     **while** $i \leq$ no of task available at time t **do**
16:         **if** $\sum_{j=p}^{j=k} w_{ij} = \delta_1 < \delta$ and $\sum_{j=p}^{j=k+1} w_{ij} > \delta$ **then**
17:             $i = i + 1$
18:             **Remove** $w_{im}, \cdots, w_{ik}$ from $\Omega(t)$
19:         **end if**
20:     **end while**
21:     **return** $w_{ip}, \cdots, w_{ik}, w_{(i+1)p'}, \cdots$
22: **end function**
23: **function** FIND_LAYERS_TO_SEND($\Omega(t)$)
24:     **while** $\Omega(t) \neq NULL$ **do**
25:         S =FIND_TRANSITION_OF_LAYERS($\Omega(t)$) ▹ *See Line 13 for definition*
26:         Check schedulability conditions (e.g., Lemma 5.2 for EDF and Lemma 5.3 for RM)
27:         **if** Schedulable **then**
28:             Continue
29:         **else**
30:             **break** ▹ *Taskset is not schedulable*
31:         **end if**
32:         **if** $t \geq T_{hyp}$ **then** ▹ *T_{hyp} is the hyperperiod of T*
33:             **break**
34:         **end if**
35:     **end while**
36:     **return** S
37: **end function**

maximum number of layers that TEE can support to reduce the SMC context switch overheads. For a given DNN task τ_i, the worst-case execution time of the model inside TEE is C_i^a, where $C_i^a = \sum_{j=1}^{j=L} C_{ij}^a$ and L_i is the number of layers. If there is L_i layers in task τ_i, then the size of each layer will be $w_{i1}, w_{i2}, ...w_{iL_i}$, where $\sum_{j=1}^{j=L_i} w_{ij} = W_i$. We first check if the following condition holds: $(w_{i1} + w_{i2}) < \delta$. We find the maximum value of k where $\sum_{j=1}^{j=k} w_{ij} = \hat{\delta} < \delta$, $\sum_{j=1}^{j=k+1} w_{ij} > \delta$. If some extra capacity is left (i.e., $\delta - \hat{\delta}$), we check the subsequent task to fit within this extra space. We find the maximum value of k for the next task where $\sum_{j=1}^{j=k} w_{(i+1)j} < (\delta - \hat{\delta})$, $\sum_{j=1}^{j=k+1} w_{(i+1)j} > (\delta - \hat{\delta})$. We check all available candidate tasks at a given time t to check whether layers can fit inside the enclave. Once we obtain the schedule profile, we repeat the same steps for all subsequent task arrivals.

Algorithm 5 formally presents the fusion approach. The fusion decision will be made when a task *(a)* arrives, *(b)* completes, or *(c)* returns from the enclave. Since the scheduler keeps track of the ready-queue and SMC returns (for example, OP-TEE tear-down APIs `TEEC_CloseSession()` and `TEEC_FinalizeContext()`), we know when to perform fusion decisions. For each scheduling decision event, the scheduler picks the fuse candidates (for instance, the loops in Line 8-Line 11, Algorithm 5). Let $\Omega(t)$ be the set of all tasks scheduled by using the vanilla EDF/RM (i.e., without any TEEs) algorithm at any given time t. We first calculate the hyperperiod of the taskset (Line 6). From $\Omega(t)$, we find the set of layers S to send to TEE (Line 9). We find the transition point k for each task and remove layers p to k from $\Omega(t)$, where p is an integer initialized to 0 (Line 18). Then, we calculate the corresponding candidate by following the condition (Line 16). We repeat this for all subsequent tasks available at that time using the `while` loop (Line 15-20). We return all the layers $w_{ip}, \cdots, w_{ik}, w_{(i+1)p'}, \cdots$ (Line 21) to S that is finding the set of layers to send to TEE (Line 9). We then check the schedulability condition (see Lemma 5.2 and 5.3 for a formal derivation). If the task is schedulable, we continue to find the next candidate to send to TEE and repeat this process till hyperperiod. In the following example, we demonstrate our proposed idea.

Example 4. Let us assume we have three tasks τ_1, τ_2, τ_3 each having 5 layers and $\delta = 7$. The size of τ_1 and τ_2 is 10, and the size of τ_3 is 5 units. We consider the size of each layer to be the same for simplicity. We cannot execute all the layers of τ_1 inside the enclave as the size of $\tau_1 > \delta$. If we execute layer-by-layer, we need five SMC switching from the normal world for five layers for each task τ_1, τ_2, and τ_3. If we send multiple layers of τ_1 that can be supported by TEE, it still requires two SMC switching i.e., $\{w_{11}, w_{12}, w_{13}\}, \{w_{14}, w_{15}\}$. For task τ_2, we also need two SMC switching $\{w_{21}, w_{22}, w_{23}\}, \{w_{24}, w_{25}\}$. For task τ_3, we need one SMC contetxt switching $\{w_{31}, w_{32}, w_{33}, w_{34}, w_{35}\}$. Hence, we need fifteen SMC switches for layer-by-layer operations to execute these three tasks. In contrast, it is possible to perform the same objective using only five SMC switches if we can send it by multiple layers. If we send multiple layers of τ_1, we still have some extra capacity left ($\delta - \delta_1 = 1$). In this case, we check whether if it is feasible to use that space capacity. In this example, $w_{11} + w_{12} + w_{13} + w_{31} = 7 \leq \delta$. Hence, we can fuse the first three layers from τ_1 and the first layer from τ_3, and then send them together to the enclave. If we repeat the same operations for the rest of the layers we get the following pattern: $\{w_{11}, w_{12}, w_{13}, w_{31}\}$,$\{w_{14}, w_{15}, w_{21}, w_{32}\}$, $\{w_{22}, w_{23}, w_{24}, w_{33}\}$,$\{w_{25}, w_{34}, w_{35}\}$ i.e., we only need four SMC switches.

5.2 Schedulabilty Conditions and Overhead Analysis

Recall that, a taskset is schedulable *(a)* if $\forall t, U(t) < 1$ and $h(t) \leq t$ (for EDF) or *(b)* if $R_i \leq D_i, \forall \tau_i$ (for RM). We now derive the expressions for $U(t)$, $h(t)$, and R_i.

Lemma 5.1. *Let $n_i^s(t)$ is the number of context switches by applying fusion for a window of duration t. System utilization $U(t)$ for a given taskset at any given time t is given by*

$$U(t) = \sum_{i=1}^{i=n} \left(\frac{\lfloor \frac{t}{T_i} \rfloor \times C_i}{t} - \frac{n_i^s(t) \times C_i^{cs}}{t} \right). \tag{3}$$

Proof. To determine the system utilization for a given taskset, we assume that each task arrives at time $t = 0$. We then calculate the number of occurrences of each task within time t using the expression $\lfloor \frac{t}{T_i} \rfloor$, where T_i represents the period of task τ_i. The overhead of each task is then given by $\lfloor \frac{t}{T_i} \rfloor \times C_i$, where C_i represents the computation time required for task τ_i. At any given time t, system utilization is $\sum_{i=1}^{i=n} \frac{\lfloor \frac{t}{T_i} \rfloor \times C_i}{t}$. However, by applying layer fusion, we can reduce context switching overhead as $\frac{n_i^s(t) \times C_i^{cs}}{t}$, where $n_i^s(t)$ is the number of context switches by applying fusion for a window of duration t. Hence, we can calculate the system utilization at any given time t as follows: $U(t) = \sum_{i=1}^{i=n} \left(\frac{\lfloor \frac{t}{T_i} \rfloor \times C_i}{t} - \frac{n_i^s(t) \times C_i^{cs}}{t} \right)$ □

Lemma 5.2 (EDF Schedulability). *The task set* Γ *is schedulable by an EDF scheduler if* $\forall t > 0$, $t < T_{max}$; $h(t) + b(t) \leq t$, $U(t) < 1$, *where*

$$h(t) = \sum_{i=1}^{i=n} \left(\lfloor \frac{t}{T_i} \rfloor C_i - \frac{n_i^s(t) \times C_i^{cs}}{t} \right). \tag{4}$$

Proof. The demand function $h(t)$ calculates the maximum execution time required by all tasks that have both their arrival times and their deadlines in a contiguous interval of length t. Recall that, $h(t)$ is given by $h(t) = \sum_{i=1}^{i=n} \lfloor \frac{t}{T_i} \rfloor C_i$. With fusion, we can reduce up to $n_i^s(t)$ context switches for each task τ_i for a window of size t. Considering this reduction, we now rewrite $h(t)$ as: $h(t) = \sum_{i=1}^{i=n} \left(\lfloor \frac{t}{T_i} \rfloor C_i - \frac{n_i^s(t) \times C_i^{cs}}{t} \right)$. □

Lemma 5.3 (RM Schedulability). *The task set* Γ *is schedulable following RM scheduling policy if* $\forall \tau_i, R_i \leq D_i$, *where* $R_i(k+1) = b_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i(k)}{T_J} \right\rceil C_j - \sum_{i=1}^{i=i} n_i^s \times C_i^{cs}$.

Proof. The response time R_i of a task τ_i is calculated iteratively using the following equation: $R_i(k+1) = b_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i(k)}{T_J} \right\rceil C_j$. For Fusion-RM, the computing time is given by $C_i = C_i^a + n_i^{cs} \times C_i^{cs}$, where n_i^{cs} is the total SMC context switches. Let us rewrite $R_i(k)$ as follows: $R_i(k+1) = b_i + (C_i^a + n_i^{cs} \times C_i^{cs}) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(k)}{T_J} \right\rceil (C_j^a + n_j^{cs} \times C_j^{cs})$. Layer fusion can reduce up to n_i^s context switches for each task τ_i. Hence, $R_i(k+1) = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j(k)}{T_J} \right\rceil C_j - \sum_{i=1}^{i=i} n_i^s \times C_i^{cs}$. □

We now calculate the reduction in SMC context switch counts when we use layer fusion.

Lemma 5.4. *If we have* z *fused tasks in* Γ, *then the total context switch reduction within the hyperperiod is* $\sum_{j=1}^{j=z}(k_j - 1)C_j^{cs}$, *where* k_j *is the number of fused layers in* j^{th} *fused task,* C_j^{cs} *is the context switch overhead.*

Proof. If we can fuse k layers from different tasks that are available at time t, then j^{th} fused task τ_j^{fused} is defined as $(C_j^{fused}, n_j^{fused})$, where C_j^{fused} is the execution time of fused task and n_j^{fused} is the SMC context switching reduction due to j^{th} fused task. If we can fuse k_j layers, then C_j^{fused} can be measured using the following equation: $C_j^{fused} = C_j^{cs} + \sum_{i=1}^{i=k_j} C_{ji}^a$, where C_{ji}^a is the computation time at i^{th} layer. If we can fuse k layers in j^{th} fused task, we can reduce $n_j^{fused} = (k_j - 1) \times C_j^{cs}$ context switches. If we have z number of fused tasks within the hyperperiod, we can define the total context switching overhead reduction as:

$$n^s = \sum_{j=1}^{j=z} n_j^s = \sum_{j=1}^{j=z} (k_j - 1) C_j^{cs}. \tag{5}$$

□

6 Evaluation

We evaluate our techniques on two fronts: *(a)* design-space exploration with various DNN workloads for EDF and RM schedulers (Section 6.1) and *(b)* case study with a UAV autopilot system (Section 6.2).

6.1 Design-Space Exploration with Deep Learning Workloads

6.1.1 Simulation Setup. We evaluate the performance of the proposed schemes using synthetically generated workloads, with parameters similar to that used in prior work [38]. We vary the system

Table 5. Systems and Workloads.

Parameters	Description
Hardware	4x ARM Cortex A53, 1 GB RAM (Raspberry Pi 3 Model B)
Rich OS	Linux 6.2.0
Trusted OS	OP-TEE 3.19.0
Workloads	• AlexNet-squeezed (Image Processing) • Tiny Darknet and YOLOv3-tiny (Object Detection) • Random: weights and run times are generated randomly

Table 6. Simulation Parameters.

Parameters	Value
Enclave capacity, δ	8 MB
Utilization, U	0%-100%
Period T	[50, 1000]
Number of layers, L	[5, 24]
Weight, W	[0.01, 7]
Execution time inside TEE per layer, c_{ij}^a	[0.1, 8]
SMC overhead, $c_s^{st} + c_s^d$	20 ms
Number of tasks, n	[5, 25]
Number of taskset for each utilization, N_u	200

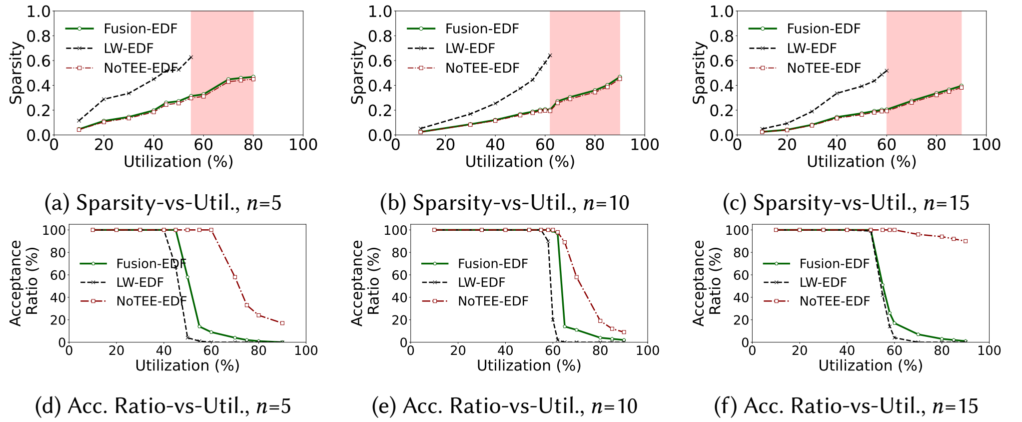
utilization from 0% to 100%. For each system utilization u in the range $[0, 10, \cdots, 100]\%$, we generate 200 tasksets, each taskset containing 5 to 15 tasks. Task periods are randomly selected from 50 to 1000. For the deep learning workload, we used three popular DNN architectures: AlexNet-squeezed [32], Tiny Darknet [52], and YOLOv3-tiny [5]. We also tested with a "random workload" where we randomly generated the number of layers, task period, size of layers, and computation time. We tested with two enclave capacities (δ): 8 MB for AlexNet-squeezed and Tiny Darknet and 16 MB for YOLOv3-tiny. We note that OP-TEE uses enclaves of similar sizes. Unless otherwise specified, we consider SMC context switch overhead ($c_s^{st} + c_s^d$) to be 20 ms. Table 5 summarizes platform and workload, and Table 6 lists key simulation parameters.

6.1.2 Schemes and Metrics. We compare layer fusion (i.e., Fusion-EDF, and Fusion-RM) with layer-wise execution technique (i.e., LW-EDF and LW-RM). For completeness, we also study a "non-secure" variant that does not consider any enclave. The schemes used in our evaluation are listed below.

- **LW-EDF** and **LW-RM**: Sends the layers sequentially (i.e., layer-wise) to the enclave using EDF (Section 4.2) and RM (Section 4.3) scheduling, respectively.
- **Fusion-EDF** and **Fusion-RM**: Groups multiple layers from multiple tasks following EDF and RM scheduling, respectively (Section 5).
- **NoTEE-EDF** and **NoTEE-RM**: DNN task execution *without* any enclave. The tasks follow the EDF (NoTEE-EDF) or RM (NoTEE-RM) scheduling policy. In this case, model confidentiality is not enforced.

We tested the above schemes with the following two metrics.

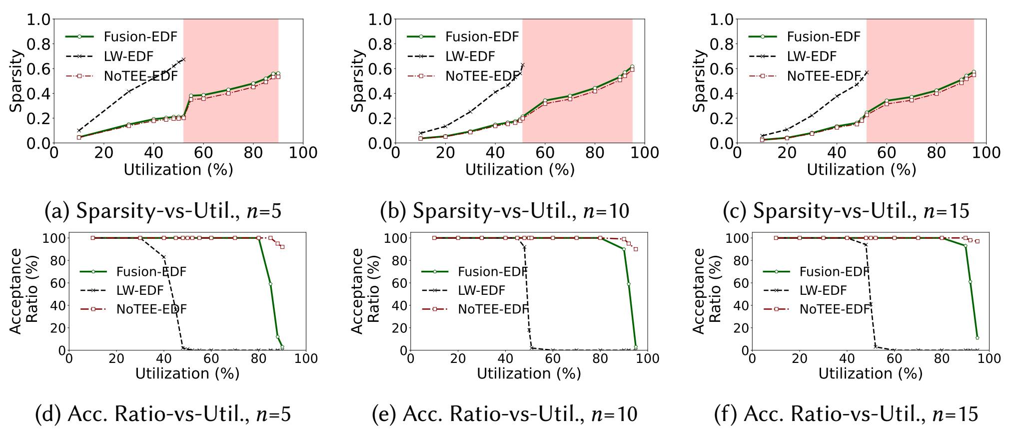
- **Sparsity**: Our newly introduced metric that shows the "spread" of the task (viz., the ratio between response time and period) [11]. Higher sparsity means tasks are completed late, and



(a) Sparsity-vs-Util., n=5 (b) Sparsity-vs-Util., n=10 (c) Sparsity-vs-Util., n=15

(d) Acc. Ratio-vs-Util., n=5 (e) Acc. Ratio-vs-Util., n=10 (f) Acc. Ratio-vs-Util., n=15

Fig. 4. Sparsity and Acceptance Ratio with varying system utilization for $\{5, 10, 15\}$ tasks using AlexNet-squeezed [32] architecture following EDF scheduling. The red shaded regions show cases where LW-EDF cannot find schedulable tasksets while other schemes can. Fusion-EDF result in better schedulability compared to LW-EDF as the utilization increases with performance penalty (i.e., both Sparsity and Acceptance Ratio are close to the No-TEE case.



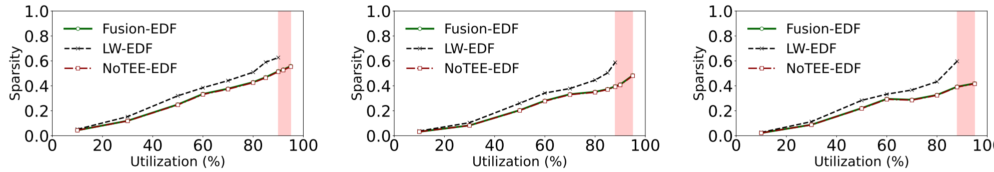
(a) Sparsity-vs-Util., n=5 (b) Sparsity-vs-Util., n=10 (c) Sparsity-vs-Util., n=15

(d) Acc. Ratio-vs-Util., n=5 (e) Acc. Ratio-vs-Util., n=10 (f) Acc. Ratio-vs-Util., n=15

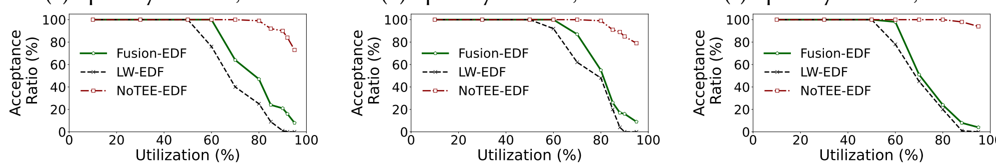
Fig. 5. Sparsity and Acceptance Ratio using Tiny Darknet [52] architecture following EDF scheduling using a setup identical to that of Fig. 4. The findings are similar.

that may result in poorer performance in terms of the DNN inference process. A Sparsity value > 1 implies the task misses the deadline.
- **Acceptance Ratio**: A commonly used metric by the real-time community that represents the fraction of tasks that meet deadlines over the total generated ones.

6.1.3 Results. We first show the Sparsity and Acceptance Ratio for varying numbers of tasks ($n = 5$, $n = 10$, and $n = 15$) for the DNN workloads listed in Table 5. The x-axis of Fig. 4 shows the various taskset utilization for randomly generated taskset running AlexNet-squeezed architecture and scheduled by EDF policy. The y-axis of Fig. 4a and Fig. 4d shows Sparsity and Acceptance Ratio, respectively. We show the Sparsity and Acceptance Ratio for Fusion-EDF (Green), LW-EDF

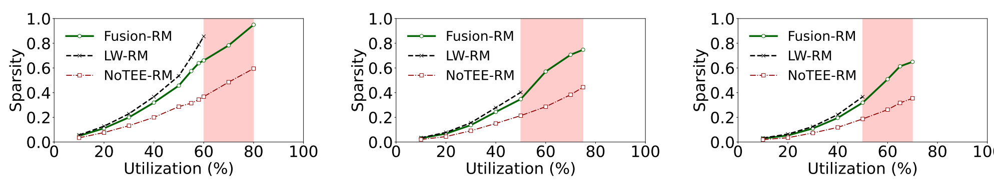


(a) Sparsity-vs-Util., n=5 (b) Sparsity-vs-Util., n=10 (c) Sparsity-vs-Util., n=15

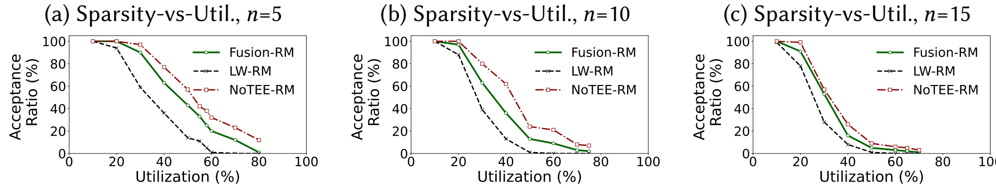


(d) Acc. Ratio-vs-Util., n=5 (e) Acc. Ratio-vs-Util., n=10 (f) Acc. Ratio-vs-Util., n=15

Fig. 6. Sparsity and Acceptance Ratio using YOLOv3-tiny [5] architecture following EDF scheduling using a setup identical to that of Fig. 4. Our findings are similar to Fig. 4 and Fig. 5.



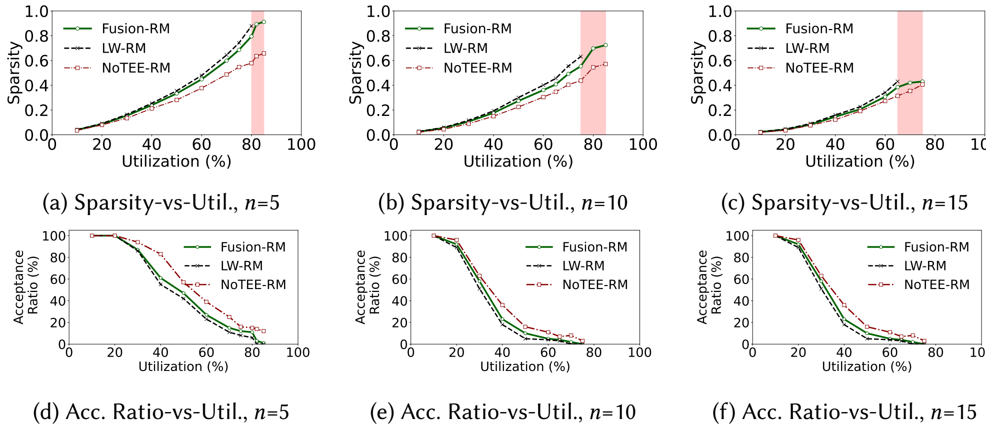
(a) Sparsity-vs-Util., n=5 (b) Sparsity-vs-Util., n=10 (c) Sparsity-vs-Util., n=15



(d) Acc. Ratio-vs-Util., n=5 (e) Acc. Ratio-vs-Util., n=10 (f) Acc. Ratio-vs-Util., n=15

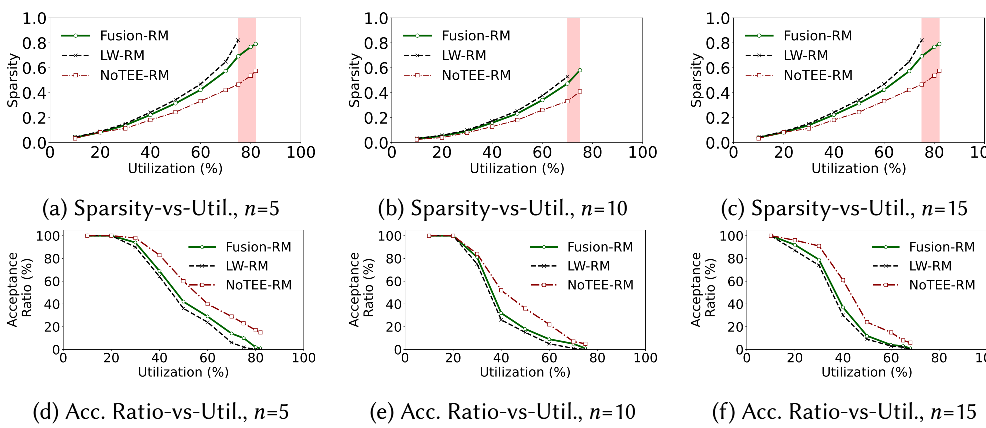
Fig. 7. Sparsity and Acceptance Ratio using AlexNet-squeezed [32] architecture for RM scheduler using a setup identical to that of Fig. 4. The findings are similar to the EDF case.

(Black), and NoTEE-EDF (Red) schemes. The red shaded regions in the figure represent the cases where LW-EDF is unable to find any schedulable candidate while Fusion-EDF finds some. For lower utilization, all schemes show similar behavior. However, Fusion-EDF outperforms LW-EDF up to 3x as the utilization increases (i.e., LW-EDF is unable to find schedulable tasksets as the utilization reaches 60%). This is expected because layer-wise execution in LW-EDF increases delay due to additional context switches. At higher utilization, that causes more tasks to miss deadlines and results in lower acceptance. We also note that the performance of fusion (both in terms of Sparsity and Acceptance Ratio) is close to NoTEE-EDF case (recall: NoTEE-EDF does not provide model confidentiality). Hence, Fusion-EDF can improve the security posture of the DNN tasks without significant overhead since it performs close to the vanilla execution (NoTEE-EDF) that does not have TEE support. In Fig. 5 and Fig. 6, we repeat the experiments with Tiny Darknet and YOLOv3-tiny architectures, respectively, and obtain similar results. We further run the experiments with the same DNN workloads for the RM scheduler; see Fig. 7 (AlexNet-squeezed), Fig. 8 (Tiny



(a) Sparsity-vs-Util., n=5
(b) Sparsity-vs-Util., n=10
(c) Sparsity-vs-Util., n=15
(d) Acc. Ratio-vs-Util., n=5
(e) Acc. Ratio-vs-Util., n=10
(f) Acc. Ratio-vs-Util., n=15

Fig. 8. Sparsity and Acceptance Ratio using Tiny Darknet [52] architecture for RM scheduler.

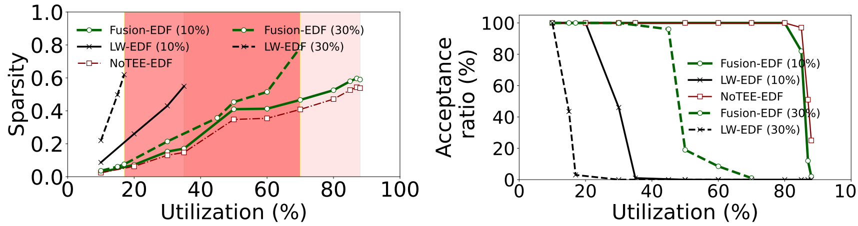


(a) Sparsity-vs-Util., n=5
(b) Sparsity-vs-Util., n=10
(c) Sparsity-vs-Util., n=15
(d) Acc. Ratio-vs-Util., n=5
(e) Acc. Ratio-vs-Util., n=10
(f) Acc. Ratio-vs-Util., n=15

Fig. 9. Sparsity and Acceptance Ratio using YOLOv3-tiny [5] architecture for RM scheduler.

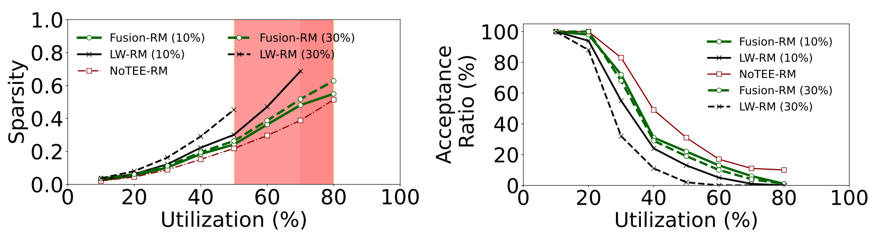
Darknet), and Fig. 9 (YOLOv3-tiny), where the data points are as follows: Fusion-RM (Green), LW-RM (Black), and NoTEE-RM (Red). The overall findings are similar to those of the EDF case. As the number of tasks increases (i.e., n = 15), we see a higher impact of context switches. As a result, Acceptance Ratio in NoTEE-EDF (NoTEE-RM) case significantly outperforms Fusion-EDF/LW-EDF (Fusion-RM/LW-RM) in highly utilized systems. However, Fusion-EDF (Fusion-RM) always results in lower Sparsity (and hence, better Acceptance Ratio) than LW-EDF (LW-RM).

To further analyze the effect of context switches on Sparsity and Acceptance Ratio, we vary the SMC overheads as a percentage of WCET. Let max($WCET$) denote the maximum WCET value observed in our experiments. The solid lines in Fig. 10 show the context switch cost as 10% of max($WCET$) values of all tasks, while dotted lines are generated with SMC overheads with 30% of max($WCET$). As the figures show, the effect of higher context switch costs causes LW-EDF to perform poorly as delays accumulating by higher context switch duration lead to longer response times (higher Sparsity values), which in turn cause more tasks to miss their deadlines (result in



(a) Sparsity-vs-Util. (b) Acc. Ratio-vs-Util.

Fig. 10. Sparsity and Acceptance Ratio for a randomly generated workload with two different context switch overheads for EDF scheduler: *(a)* 10% of max(WCET) values (solid lines) and *(b)* 30% of max(WCET) values (dotted lines). Larger context switch delays result in higher Sparsity for LW-EDF when compared to Fusion-EDF, which in turn, reduces the percentage of schedulable tasksets. Fusion-EDF performs identically to No-TEE due to lower context switch delays.



(a) Sparsity-vs-Util. (b) Acc. Ratio-vs-Util.

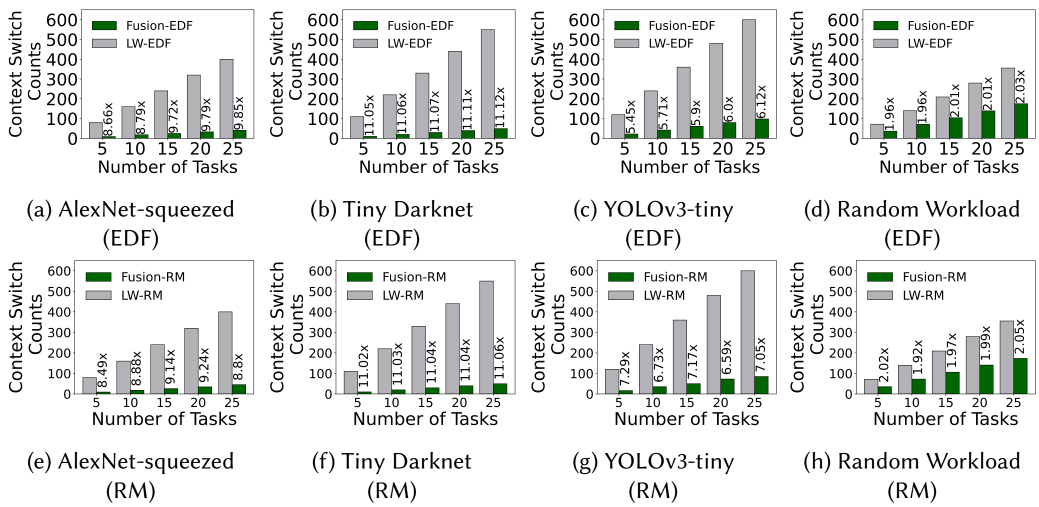
Fig. 11. Sparsity and Acceptance Ratio for a randomly generated workload with two different context switch overheads following RM scheduling using a setup identical to that of Fig. 10. The findings are similar to the EDF case.

lower Acceptance Ratio). In Fig. 11, we repeat the experiments with an identical setup but for the RM scheduler. We see a similar performance trend for RM to that observed for EDF.

Fusion-EDF (Fusion-RM) outperforms LW-EDF (LW-RM), especially for high utilization scenarios. Further, the overhead of layer fusion is negligible as its performance is close to the vanilla execution (e.g., NoTEE-EDF/NoTEE-RM). Systems with longer TEE context switch delay can be significantly benefited by layer fusion compared to the layer-wise partitioning.

In the next set of experiments we measure the number of SMC context switches as follows: LW-EDF vs. Fusion-EDF (Fig. 12a-Fig.12d) and LW-RM vs. Fusion-RM (Fig. 12e-Fig.12h). For these experiments, we set the system utilization to 50%. Note that, as NoTEE-EDF and NoTEE-RM do not have any enclave, there are no SMC calls (context switches). Hence, our plots exclude NoTEE-EDF/NoTEE-RM in this case. As the figures show, layer fusion can significantly reduce context switch counts compared to layer-wise execution (5.45x-11.1x for EDF and 6.59x-11.06x for RM) for all three architectures. This is because Fusion-EDF/Fusion-RM groups multiple layers; hence, overall, the number of SMC calls is reduced.

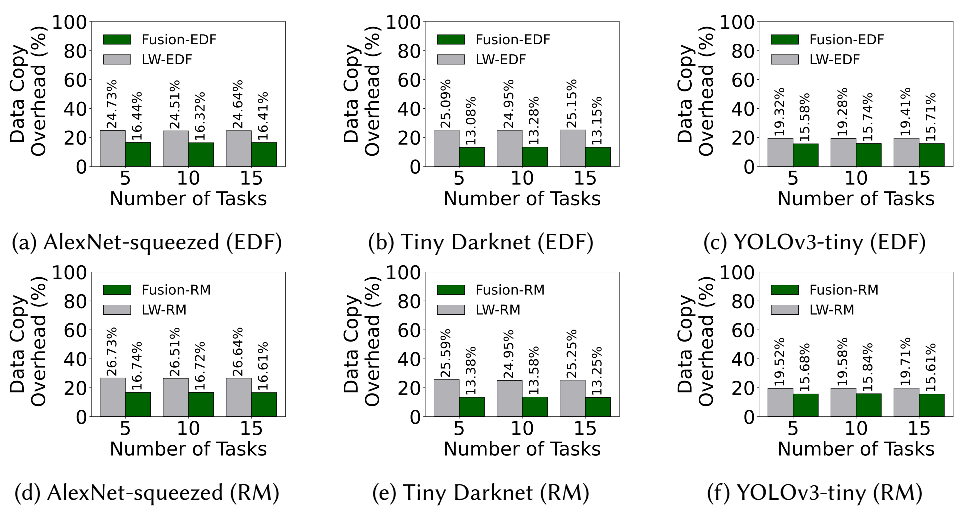
Layer fusin significantly reduces the number context switches (1.96x-11.12x for EDF and 1.92x-11.06x for RM, see Fig. 12). This reduction of context switches also contributes to a higher schedulability (see Fig. 4-Fig. 9).



(a) AlexNet-squeezed (EDF) (b) Tiny Darknet (EDF) (c) YOLOv3-tiny (EDF) (d) Random Workload (EDF)

(e) AlexNet-squeezed (RM) (f) Tiny Darknet (RM) (g) YOLOv3-tiny (RM) (h) Random Workload (RM)

Fig. 12. Context switch overhead comparison for three known architectures (e.g., Tiny Darknet, AlexNet-squeezed, YOLOv3-tiny) and one for a random workload. Layer fusion reduces context switch overheads compared to layer-wise partitioning (1.96x-11.12x for EDF and 1.92x-11.06x for RM).

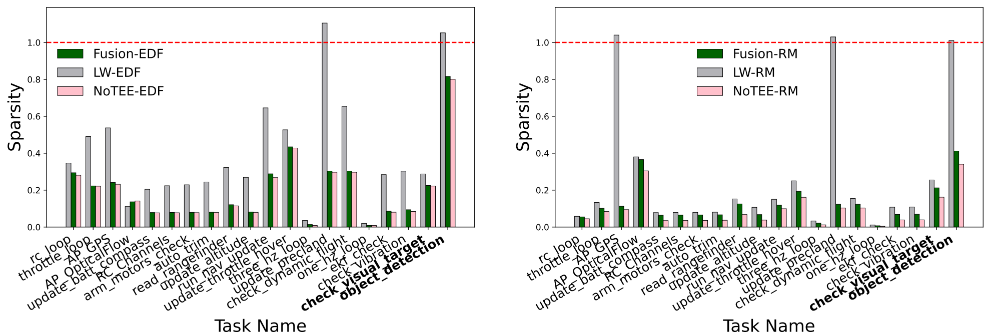


(a) AlexNet-squeezed (EDF) (b) Tiny Darknet (EDF) (c) YOLOv3-tiny (EDF)

(d) AlexNet-squeezed (RM) (e) Tiny Darknet (RM) (f) YOLOv3-tiny (RM)

Fig. 13. Data copy overheads for various DNN workloads. Inference confidentiality increases response times due to additional data transfers and SMC calls. However, this overhead remains constant with the increasing number of tasks. The overheads of Fusion-EDF (Fusion-RM) are less compared to LW-EDF (LW-RM) due to the fewer context switches.

Recall from Section 3.1 that each time a context switch is performed, normal world (encrypted) data needs to be transferred to the secure world. We now analyze this data copy overhead. The experiments in Fig. 13a-13f show the overheads for the various DNN workloads (AlexNet-squeezed, Tiny Darknet, and YOLOv3-tiny) and a varying number of tasks ($n = 5$, $n = 10$, and $n = 15$) for EDF and RM schedulers running on Raspberry Pi and OP-TEE. To calculate the end-to-end data copy overheads, we first measured the response times for NoTEE-EDF/NoTEE-RM case and



(a) (b)

Fig. 14. Sparsity for ArduPlilot controller tasks: *(a)* EDF (left) and *(b)* RM (right). Bold tasks are our added DNN inference workload, and the red horizontal line denotes the deadline. The increasing number of context switches in LW-EDF (LW-RM) caused a larger spread of tasks (higher Sparsity), and as a result, two (three) tasks missed deadlines. For Fusion-EDF and Fusion-RM, all tasks meet their deadlines.

then subtracted these values from the response times of each of the fusion schemes. Finally, we normalized them with the task periods (i.e., calculated Sparsity) and obtained the overhead percentage. We only considered schedulable tasksets. For each data point, we generated 100 samples and took the 90$^{\text{th}}$ percentile value. As the figure shows, enabling confidential inference comes with a cost, i.e., increase in response times. This data copy overhead is system (i.e., underlying SMC implementations) and workload (i.e., DNN layers/architecture) dependent. For instance, we find that the additional delay in response times due to transferring context for LW-EDF and Fusion-EDF are *(a)* 2.39 s and 1.34 s (AlexNet-squeezed), *(b)* 2.95 s and 1.54 s (Tiny Darknet), and *(c)* 6.96 s and 5.62 s (YOLOv3-tiny), respectively on Raspberry Pi+OP-TEE setup (recall: each SMC overhead could be as high as 20 ms; see Table 1). Likewise, additional delays in response times due to transferring data back and forth from the enclave and the normal world for LW-RM and Fusion-RM are *(a)* 2.31 s and 1.76 s (AlexNet-squeezed), *(b)* 2.89 s and 1.65 s (Tiny Darknet), and *(c)* 7.43 s and 5.86 s (YOLOv3-tiny), respectively. As the figure shows, the data copy overhead scales well with the increasing number of tasks (remains constant). Further, Fusion-EDF and Fusion-RM incur lower overheads due to a reduced number of context switches, as we also observed in prior experiments (Fig. 12).

> *Confidential deep inference comes with a cost: it increases response times due to additional data transfer between normal and secure worlds. However, this data transfer overhead does not increase significantly with the increasing number of tasks.*

6.2 Case Study with a UAV Controller System

In the final set of experiments (Fig. 14), we evaluate Fusion-EDF, LW-EDF, and NoTEE-EDF (resp. Fusion-RM, LW-RM, NoTEE-RM) with a UAV autopilot system (ArduPlilot [2]) running on Raspberry Pi 3 [54]. The ArduPlilot controller has 18 real-time tasks (defined in `/ArduCopter/Copter.cpp`). Since the vanilla controller does not have any DNN workload, we included two additional inference tasks (i.e., `check_visual_target()` and `object_detection()`) that use Tiny Darknet and YOLOv3-tiny models, respectively, to perform object detection.[2]

[2]Note: Conceptually, our added DNN inference tasks can be scheduled like other existing ArduPliot tasks, for instance, by modifying the variable `AP_Scheduler::Task Copter::scheduler_tasks[]` in `Copter.cpp` and inserting them in the

The inference tasks were invoked periodically (i.e., in 5 seconds intervals). The total system utilization (including two of the included DNN tasks) was 0.75. We selected these parameters by trial and error to ensure that we can evaluate the taskset for a highly utilized setup (i.e., above the theoretical L&L bound [45]), but at the same time, they remain schedulable at least for the base case (NoTEE-EDF/NoTEE-RM) that does not have TEE related overheads.

Each of the bars in Fig. 14 shows the various tasks and their Sparsity for each of the three schemes. The figure shows that due to high context switches, LW-EDF (LW-RM) misses deadlines for two (three) real-time tasks (i.e., Sparsity > 1). The controller tasks that miss deadlines in LW-RM are higher priority than DNN tasks. However, due to the non-preemptive execution of TEE segments, the DNN tasks cause extra inference (blocking delay), which increases response times. In contrast, both Fusion-EDF (Fusion-RM) and NoTEE-EDF (NoTEE-RM) met all deadlines. Engineers can conduct similar design-time tests to assess the feasibility of adapting confidential DNN techniques in their target system.

High SMC context switch overheads cause LW-EDF and LW-RM to miss a few deadlines; Fusion-EDF and Fusion-RM, in contrast, were able to meet all deadlines.

7 Discussion

Confidentiality-Schedulability Trade-Offs. In this work, we assume that all layers run inside TEEs using EDF and RM scheduling. There are use cases in which not all layers require confidentiality. For example, in image/voice recognition applications where the user may prefer not to reveal input and processed data, running initial input and final output layers within TEE should suffice. Our future research will look into the variable number of TEE executions and the performance trade-offs in a real-time context.

Further Optimization using Sub-Layer Partitioning. Our current design currently selects a whole slice of a layer and fuses it with another task. For example, consider τ_i has four layers $\{l_i^{11}, \cdots, l_i^{14}\}$ and τ_j has three layers $\{l_j^{11}, \cdots, l_j^{13}\}$. When feasible (for instance, the enclave can fit layers), we fuse all seven layers. It could also be possible to obtain a "partial" slice of a layer in case a complete slice does not fit in the enclave (or the enclave has a little extra capacity). For instance, in the example above, $\{l_i^{11}, l_i^{12}, l_j^{12}, l_j^{13}\}$ could form a fusion group in case all seven layers do not fit to further improve schedulability. However, extending this paper to incorporate such splitting needs further research.

Mixed Workload. We assume only inference tasks use TEEs. In practice, other (non-DNN) tasks could also use TEEs, thus potentially limiting enclave availability. Our proposed idea can be extended for such scenarios considering extra *slack* reclaimed from other non-inference tasks.

Limiting Schedule Observability. The overall security of our confidentiality-preserving inference technique relies on the underlying TEE architecture (TrustZone). However, TrustZone could also be vulnerable, especially exposed to schedule-based attacks [6] for real-time context. One approach to limit such observability is to introduce "noise" in the scheduler [18]. For instance, instead of fusing the same set of tasks, we can select fusion candidates from different groups, thus limiting the predictability and, hence, reducing the chances of information leakage. However, this may

scheduler through the `SCHED_TASK()` function. However, to the best of our knowledge, ArduPliot does not support TEE, and there is no DNN library compatibility in the existing scheduler implementation. We use ArduPilot parameters to demonstrate the tradeoffs of adding TEE-based DNN inference in a realistic setup (e.g., UAV autopilot system). Adapting ArduPliot for TEEs (say OP-TEE) and adding library support for DNN requires a significant redesign, and we leave this for future exploration.

cause priority inversions for some tasks, and we need a new schedulability analysis to ensure that temporal constraints are met.

Compatibility with Other TEEs. Our work focuses on TrustZone, as ARM is widely used for embedded and cyber-physical application development. Conceptually similar ideas can be adopted for other TEEs, such as Intel SGX. In SGX, switching between enclave mode and regular execution is performed through the ECALL (enclave call) and OCALL (outside call) mechanisms. An ECALL allows an application to execute secure operations inside an enclave by securely switching the execution context from normal mode to enclave mode. An OCALL enables an enclave to invoke external untrusted services, such as I/O or file system access. The SMC instructions in TrustZone for switching between the normal and secure world are conceptually similar to SGX's ECALL/OCALL mechanisms. Despite conceptual similarities, SGX (x86) and TrustZone (ARM) are different architectures. A complete porting of our proposed ideas to other SGX or other TEEs needs further investigation, and we leave this for future research.

8 Related Work

In early work [10], we survey existing confidential deep learning techniques to find that adapting confidentiality for the DNN inference process for real-time cyber-physical systems is still in the early stages. The closest line of research is our prior work [11] — which is a follow-up on our preliminary investigation presented at a workshop [9] — where we show how to integrate confidential DNN techniques for EDF schedulers. This paper extends our prior work to fixed-priority (RM) schedulers.

AegisDNN [63] proposes to execute only a few layers that will be executed inside SGX-based TEEs. However, AegisDNN is primarily designed for soft real-time systems and allows for missed deadlines. Our work aims to provide hard real-time guarantees. SuperTEE [50] aims to reduce TEE task switching overhead. However, SuperTEE is not designed for learning-enabled real-time systems and can not be directly adapted for multi-layer DNN tasks. Researchers also propose various techniques (e.g., Subflow [38], AppNet [12], Zygarde [33], LaLaRAND [35]) to make deep learning "time-aware," but they do not consider trusted execution or model confidentiality aspects.

There exists other work for general-purpose systems. DarkneTZ [48] proposes to execute only a few layers that will be executed inside TEE, which is not suitable for applications that require executing all layers within TEE. Layers that execute outside of the secure world expose information to the untrusted normal world, raising data privacy concerns. A similar line of work exists (e.g., HybridTEE [27], Confidential DL [62], Occlumency [39], SecureQNN [21]), for executing machine learning workloads inside TEEs. However, none of them consider real-time constraints. The proposed research is one of the fundamental works that explores time-aware confidential DNN inference techniques for learning-enabled real-time systems.

9 Conclusion

This study presents novel techniques to ensure real-time guarantees for confidential deep neural inference. We demonstrate how to partition large DNN models and schedule them using two key real-time scheduling policies (EDF and RM). Additionally, we introduce an optimization strategy (layer fusion) to reduce the overhead caused by frequent TEE context switching. The techniques developed in this work provide a framework for engineers of autonomous systems to analyze the performance trade-offs in terms of overhead and scheduling efficiency while integrating confidential DNN workloads.

Acknowledgments

This research is partly supported by the US National Science Foundation Award 2312006 and Washington State University Grant PG00021441. Any findings, opinions, recommendations, or conclusions expressed in this paper are solely those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Apache NuttX. https://nuttx.apache.org.

[2] Ardupilot project. https://github.com/ArduPilot/ardupilot.

[3] FreeRTOS. https://www.freertos.org.

[4] Intel software guard extensions: Intel SGX SDK for Linux OS. http://intel.com. accessed: 2020-06-30.

[5] Adarsh, P., Rathi, P., and Kumar, M. Yolo v3-Tiny: Object detection and recognition using one stage improved model. In *2020 6th international conference on advanced computing and communication systems (ICACCS)* (2020), IEEE, pp. 687–694.

[6] Aguida, M. A., and Hasan, M. Work in progress: Exploring schedule-based side-channels in TrustZone-enabled real-time systems. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2022), IEEE, pp. 301–304.

[7] Alves, T. TrustZone: Integrated hardware and software security. *Information Quarterly 3* (2004), 18–24.

[8] Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes 24*, 2 (1991), 127–132.

[9] Babar, M. F., and Hasan, M. Real-time scheduling of TrustZone-enabled DNN workloads. In *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy* (2022), pp. 63–69.

[10] Babar, M. F., and Hasan, M. Trusted deep neural execution—a survey. *IEEE Access* (2023).

[11] Babar, M. F., and Hasan, M. DeepTrust^{RT}: Confidential deep neural inference meets real-time! In *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)* (2024), Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[12] Bateni, S., and Liu, C. ApNet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)* (2018), IEEE, pp. 67–79.

[13] Benkraouda, H., and Nahrstedt, K. Image reconstruction attacks on distributed machine learning models. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning* (2021), pp. 29–35.

[14] Burns, A., and Wellings, A. J. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[15] Buttazzo, G. C. Rate monotonic vs. edf: Judgment day. *Real-Time Systems 29* (2005), 5–26.

[16] Buttazzo, G. C., and Buttanzo, G. *Hard real-time computing systems*, vol. 356. Springer, 1997.

[17] Buttazzo, G. C., and Buttanzo, G. *Hard real-time computing systems*, vol. 356. Springer, 1997.

[18] Chen, C.-Y., Sanyal, D., and Mohan, S. Indistinguishability prevents scheduler side channels in real-time systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), pp. 666–684.

[19] Chen, J.-J. Partitioned multiprocessor fixed-priority scheduling of sporadic real-time tasks. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)* (2016), IEEE, pp. 251–261.

[20] Chen, Z., Li, G., Pattabiraman, K., and DeBardeleben, N. BinFI: An efficient fault injector for safety-critical machine learning systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), pp. 1–23.

[21] Costa, M., Gomes, T., Cabral, J., Monteiro, J., Tavares, A., and Pinto, S. Secureqnn: Introducing a privacy-preserving framework for qnns at the deep edge. In *International Conference on Data Science and Artificial Intelligence* (2023), Springer, pp. 3–17.

[22] Davis, R. I. A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review 11*, 1 (2014), 8–19.

[23] Dertouzos, M. L. Control robotics: The procedural control of physical processes. In *Proceedings IF IP Congress, 1974* (1974).

[24] Elgamal, T., and Nahrstedt, K. Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)* (2020), IEEE, pp. 519–528.

[25] Farhadi, A., and Redmon, J. Yolov3: An incremental improvement. In *Computer vision and pattern recognition* (2018), vol. 1804, Springer Berlin/Heidelberg, Germany, pp. 1–6.

[26] Farhadi, A., and Redmon, J. Yolov3: An incremental improvement. In *Computer vision and pattern recognition* (2018), vol. 1804, Springer Berlin/Heidelberg, Germany, pp. 1–6.

[27] Gangal, A., Ye, M., and Wei, S. HybridTEE: Secure mobile DNN execution using hybrid trusted execution environment. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)* (2020), IEEE, pp. 1–6.
[28] Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. SqueezeNext: Hardware-aware neural network design. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (2018), pp. 1638–1647.
[29] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[30] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[31] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems 28* (2015).
[32] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).
[33] Islam, B., and Nirjon, S. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4*, 3 (sep 2020).
[34] Islam, M. S., Zamani, M., Kim, C. H., Khan, L., and Hamlen, K. W. Confidential execution of deep learning inference at the untrusted edge with ARM TrustZone. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2023), CODASPY'23, Association for Computing Machinery, p. 153–164.
[35] Kang, W., Lee, K., Lee, J., Shin, I., and Chwa, H. S. LaLaRAND: Flexible layer-by-layer CPU/GPU scheduling for real-time dnn tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)* (2021), pp. 329–341.
[36] Kim, K., Kim, C. H., Rhee, J. J., Yu, X., Chen, H., Tian, D., and Lee, B. Vessels: Efficient and scalable deep learning prediction on trusted processors. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (2020), pp. 462–476.
[37] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 6 (2017), 84–90.
[38] Lee, S., and Nirjon, S. SubFlow: A dynamic induced-subgraph strategy toward real-time dnn inference and training. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2020), IEEE, pp. 15–29.
[39] Lee, T., Lin, Z., Pushp, S., Li, C., Liu, Y., Lee, Y., Xu, F., Xu, C., Zhang, L., and Song, J. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking* (2019), pp. 1–17.
[40] Lehoczky, J., Sha, L., and Ding, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS* (1989), vol. 89, pp. 166–171.
[41] Li, G., Pattabiraman, K., and DeBardeleben, N. TensorFI: A configurable fault injector for TensorFlow applications. In *IEEE International symposium on software reliability engineering workshops (ISSREW)* (2018), IEEE, pp. 313–320.
[42] Li, N., Qardaji, W., Su, D., Wu, Y., and Yang, W. Membership privacy: A unifying framework for privacy definitions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), pp. 889–900.
[43] Linaro. Open portable trusted execution environment. https://www.op-tee.org, Accessed on 2021.
[44] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM) 20*, 1 (1973), 46–61.
[45] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM) 20*, 1 (1973), 46–61.
[46] Liu, R., Garcia, L., Liu, Z., Ou, B., and Srivastava, M. 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* (2021), pp. 67–79.
[47] Mienye, I. D., and Swart, T. G. A comprehensive review of deep learning: Architectures, recent advances, and applications. *Information 15*, 12 (2024).
[48] Mo, F., Shamsabadi, A. S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A., and Haddadi, H. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services* (2020), pp. 161–174.
[49] Moffat, A. Huffman coding. *ACM Computing Surveys (CSUR) 52*, 4 (2019), 1–35.
[50] Mukherjee, A., Mishra, T., Chantem, T., Fisher, N., and Gerdes, R. Optimized trusted execution for hard real-time applications on COTS processors. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems* (2019), pp. 50–60.
[51] Pinto, S., and Santos, N. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR) 51*, 6 (2019), 1–36.
[52] Redmon, J. Tiny darknet: https://pjreddie.com/darknet/tiny-darknet/.
[53] Redmon, J. Darknet: Open source neural networks in C. http://pjreddie.com/darknet/, 2013–2016.
[54] Richardson, M., and Wallace, S. *Getting started with Raspberry Pi*. O'Reilly Media, Inc., 2012.

[55] Sabt, M., Achemlal, M., and Bouabdallah, A. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa* (2015), vol. 1, IEEE, pp. 57–64.
[56] Shah, D., Campbell, W., and Zulkernine, F. H. A comparative study of lstm and dnn for stock market forecasting. In *2018 IEEE international conference on big data (big data)* (2018), IEEE, pp. 4148–4155.
[57] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)* (2017), IEEE, pp. 3–18.
[58] Shrestha, A., and Mahmood, A. Review of deep learning algorithms and architectures. *IEEE Access 7* (2019), 53040–53065.
[59] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[60] Singh, A., and Baruah, S. Global EDF-based scheduling of multiple independent synchronous dataflow graphs. In *2017 IEEE Real-Time Systems Symposium (RTSS)* (2017), pp. 307–318.
[61] Stankovic, J. A., Spuri, M., Ramamritham, K., and Buttazzo, G. *Deadline scheduling for real-time systems: EDF and related algorithms*, vol. 460. Springer Science & Business Media, 1998.
[62] VanNostrand, P. M., Kyriazis, I., Cheng, M., Guo, T., and Walls, R. J. Confidential deep learning: Executing proprietary models on untrusted devices. *arXiv preprint arXiv:1908.10730* (2019).
[63] Xiang, Y., Wang, Y., Choi, H., Karimi, M., and Kim, H. AegisDNN: Dependable and timely execution of DNN tasks with SGX. In *IEEE Real-Time Systems Symposium (RTSS)* (2021), IEEE, pp. 68–81.
[64] Zhang, F., and Burns, A. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers 58*, 9 (2009), 1250–1258.