DeepTrust$^{\mathrm{RT}}$: Confidential Deep Neural Inference Meets Real-Time!

Mohammad Fakhruddin Babar
Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA

Monowar Hasan
Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA

Abstract

Deep Neural Networks (DNNs) are becoming common in "learning-enabled" time-critical applications such as autonomous driving and robotics. One approach to protect DNN inference from adversarial actions and preserve model privacy/confidentiality is to execute them within trusted *enclaves* available in modern processors. However, running DNN inference inside limited-capacity enclaves while ensuring timing guarantees is challenging due to *(a)* large size of DNN workloads and *(b)* extra switching between "normal" and "trusted" execution modes. This paper introduces new time-aware scheduling schemes – DeepTrust$^{\mathrm{RT}}$– to *securely* execute deep neural inferences for learning-enabled real-time systems. We first propose a variant of EDF (called DeepTrust$^{\mathrm{RT}}$-LW) that *slices* each DNN layer and runs them sequentially in the enclave. However, due to extra context switch overheads of individual layer slices, we further introduce a novel *layer fusion* technique (named DeepTrust$^{\mathrm{RT}}$-FUSION). Our proposed scheme provides hard real-time guarantees by *fusing* multiple layers of DNN workload from multiple tasks; thus allowing them to fit and run concurrently within the enclaves while maintaining real-time guarantees. We implemented and tested DeepTrust$^{\mathrm{RT}}$ ideas on the Raspberry Pi platform running OP-TEE+DarkNet-TZ DNN APIs and three DNN workloads (AlexNet-squeezed, Tiny Darknet, YOLOv3-tiny). Compared to the layer-wise partitioning approach (DeepTrust$^{\mathrm{RT}}$-LW), DeepTrust$^{\mathrm{RT}}$-FUSION can schedule up to 3x more tasksets and reduce context switches by up to 11.12x. We further demonstrate the efficacy of DeepTrust$^{\mathrm{RT}}$ using a flight controller (ArduPilot) case study and find that DeepTrust$^{\mathrm{RT}}$-FUSION retains real-time guarantees where DeepTrust$^{\mathrm{RT}}$-LW becomes unschedulable.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Security and privacy → Systems security

Keywords and phrases DNN, TrustZone, Real-Time Systems

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.13

Supplementary Material *Software (Source Code)*: `https://github.com/CPS2RL/DeepTrust-RT` [6]

Funding 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 the paper are those of the authors and do not necessarily reflect the sponsor's views.

1 Introduction

The emergence of modern IoT-specific applications (such as autonomous vehicles, drones, and cognitive robots, among others) coupled with advances in computing power and hardware efficiency pushed artificial intelligence toward embedded devices. Engineers in modern safety-critical applications are progressively deploying more complex deep learning models to meet the need for on-device intelligence [24]. Many safety-critical learning-enabled systems also have stringent timing (viz., "real-time") requirements. For example, an autonomous vehicle must periodically scan and recognize objects in its surroundings. This is often performed by executing a deep neural network (DNN) inference chain. Any delay in the object recognition process may jeopardize decision-making, thus threatening the safety of the vehicle, passengers, and others around it.

Executing DNN models on end-user devices introduces new security and confidentiality challenges. Since most autonomous systems collect sensitive information (e.g., location and driving dynamics, reconnaissance images, medical records), data leakage from the learning task results in privacy concerns. Further, a compromised system could redistribute proprietary models (e.g., parameters, intermediate results, final outputs), leaking the intellectual property of the model provider. For instance, researchers have demonstrated attacks such as membership inference [28, 37], fault injection [14, 27], and input reconstruction [11], which can leak private model information and cause misclassification.

One way to tackle the DNN confidentiality problem is to execute the inference tasks inside trusted enclaves such as Intel SGX [2] or ARM TrustZone [5] available in modern processors. Enabling trusted execution for DNN workloads is challenging as most DNN tasks are compute/memory-heavy and do not fit within the enclave memory. To put this in context, VGG-16 [38], an image classification task, requires 528 MB of memory for runtime computations. In contrast, OP-TEE [29], an open-source TrustZone stack, only supports 16 MB of enclave memory. To address this issue, researchers proposed various techniques to "slice" and execute DNN workload inside trusted enclaves [9]. However, they are designed for general-purpose mobile/edge computing platforms (i.e., do not provide real-time guarantees). Simply retrofitting 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, although we can split the DNN model into multiple partitions [16, 31, 35, 41], as we shall see in this paper (Section 4.2), slicing and moving inference tasks back and forth between the trusted and normal execution mode results in extra delays due to high context switch overheads. This may cause some (or all!) critical tasks to miss their deadlines [7].

Hence, our research aims to address the following problem.

The confidentiality of DNN models running on an untrusted device can be maintained by executing them within trusted enclaves.

Challenge: *How do we ensure compute-heavy real-time DNN tasks fit in limited capacity enclaves while retaining their timing guarantees (deadlines)?*

In response to the above problem, we develop scheduling models (called DeepTrust^{RT}) that ensure a set of learning-enabled real-time tasks can retain model confidentiality. Our first attempt to make the DNN inference tasks *trusted* and *time-aware* relies on a slicing mechanism that partitions DNN models *layer-by-layer* [41]. The idea is to sequentially send one DNN layer at a time to the enclave, perform its computation, and get the results back. However, a single layer may often not fit in the enclave due to its large size. Hence, we use the Deep Compression [19] to reduce DNN model size considering the enclave capacity (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. We call our first approach DeepTrust^{RT}-LW. We also derive related schedulability conditions (Section 4.2).

We find that despite real-time guarantees, DeepTrust^{RT}-LW results in poorer throughput (i.e., fewer tasks are schedulable) due to high context switch overheads introduced by the layer-by-layer partitioning technique. Hence, we further optimize DeepTrust^{RT}-LW with a novel "fusion" approach that selectively *groups multiple layers from multiple tasks*, considering enclave capacity and deadline constraints (Section 5). We name this technique

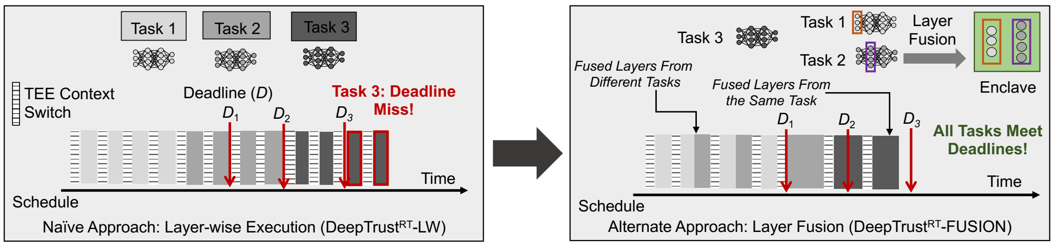

Task 1
Task 2
Task 3
TEE Context Switch
Deadline (*D*)
D$_1$
D$_2$
D$_3$
Task 3: Deadline Miss!
Time
Schedule
Naïve Approach: Layer-wise Execution (DeepTrust^{RT}-LW)
Task 3
Task 1
Task 2
Layer Fusion
Enclave
Fused Layers From Different Tasks
Fused Layers From the Same Task
D$_1$
D$_2$
D$_3$
All Tasks Meet Deadlines!
Time
Schedule
Alternate Approach: Layer Fusion (DeepTrust^{RT}-FUSION)

Figure 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. If we slice the model *layer-by-layer* to fit in the enclave and send them sequentially (left rectangle, named DeepTrust$^{\mathrm{RT}}$-LW), extra context switch overheads may violate real-time constraints. Hence, we also introduce a novel "layer fusion" technique (named DeepTrust$^{\mathrm{RT}}$-FUSION, right rectangle) that *groups multiple layers from multiple tasks* together to reduce context switch costs and results in better schedulability.

DeepTrust$^{\mathrm{RT}}$-FUSION. Figure 1 illustrates the key intuition of DeepTrust$^{\mathrm{RT}}$-FUSION for a three-task system. When DNN layers are sent sequentially to the enclave, extra context switch overheads cause longer response time, and one of the tasks misses 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 proposed layer fusion approach *(a)* better utilizes enclave capacity, and *(b)* reduces context switches (normal-to-secure and secure-to-normal) incurred by the layer-by-layer partitioning technique (up to 11.12x for a set of 25 tasks at 50% utilization, see Section 6). As a result, DeepTrust$^{\mathrm{RT}}$-FUSION achieves better schedulability compared to the DeepTrust$^{\mathrm{RT}}$-LW approach.

In this work, we focus on ARM-based enclaves (viz., TrustZone) as ARM is the dominant architecture for embedded applications. However, our ideas are general and can be adapted to other enclaves (such as SGX) without loss of generality. We tested both DeepTrust$^{\mathrm{RT}}$-LW and DeepTrust$^{\mathrm{RT}}$-FUSION on three realistic workloads (e.g., AlexNet-squeezed [21], Tiny Darknet [21], YOLOv3-tiny [3]) running on Raspberry Pi and performed design-space exploration (Section 6). We also conducted a case study using the ArduPlilot UAV autopilot system [1] and DNN-enabled workload (YOLOv3-tiny, Tiny Darknet) and found that DeepTrust$^{\mathrm{RT}}$-FUSION can meet all deadlines for critical tasks while DeepTrust$^{\mathrm{RT}}$-LW misses some (Section 6.2).

Our Contributions. This research is one of the first efforts to enable time-aware trusted DNN execution for learning-enabled real-time systems. Our contributions are as follows:

- Enabling *timing guarantees* for performing *confidential deep inference* in latency-critical learning-enabled systems.
- A new *scheduling framework and analytical model* (DeepTrust$^{\mathrm{RT}}$-LW) to determine the feasibility of deploying a given real-time DNN workload on TrustZone enclaves (Section 4).
- A *novel task fusion approach* (DeepTrust$^{\mathrm{RT}}$-FUSION) to further reduce TEE context switch overheads while retaining real-time guarantees (Section 5).

We performed thorough design-space exploration using three DNN architectures (AlexNet-squeezed, Tiny Darknet, YOLOv3-tiny). Our case study on a UAV system (ArduPliot) running on Raspberry Pi further demonstrates the efficacy and feasibility of the proposed techniques.

We now start with a background on trusted enclaves (TrustZone) and DNN architecture (Section 2) before introducing our model and related assumptions (Section 3).

2 Background

2.1 Trusted Execution and ARM TrustZone

Trusted execution environments (TEEs) offer a secure and isolated runtime environment. TEEs ensure confidentiality and integrity for the code and data that can not be exploited even if the host (i.e., main) OS is compromised. Among various off-the-shelf TEE implementations, Intel SGX [2] and ARM TrustZone [5] are two widely used technologies in many IoT/mobile applications. SGX is usually used for general-purpose computers and servers. In contrast, TrustZone architecture is more suitable for embedded applications and hence is the focus of our work.

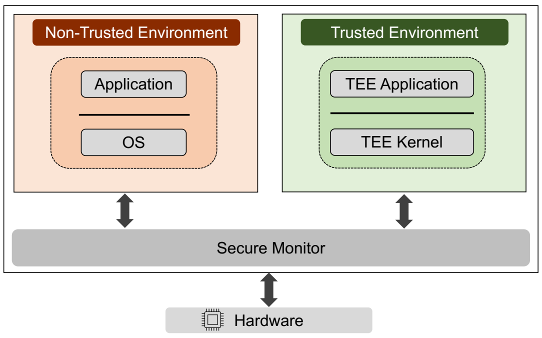


Figure 2 TrustZone architecture.

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 minimal trusted kernel (e.g., OP-TEE [29]). 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 employs a customized mechanism known as secure monitor call (SMC) to switch between these two states. When an SMC instruction is invoked in the normal world, the processor cores perform a context switch from the normal world to the secure world and pause normal world operations. As a security precaution, the normal world cannot access secure memory, while the secure world can access normal world memory. TrustZone also ensures the isolation of external peripherals.

2.2 Deep Neural Inference

DNNs comprise an input layer, one or more hidden layers, and an output layer. Each layer consists of interconnected nodes, where the connections between nodes are represented as edges, each associated with a weight, and each node has an associated threshold value. When the output of a node exceeds its threshold, the node is activated, and the data is propagated to the next layer. Figure 3 illustrates a simplified DNN architecture.

DNN algorithms establish models based on training data, enabling them to generate predictions without explicit programming. During the inference process, input data is passed through the layers, and each layer performs matrix multiplications on the data. The final layer’s outputs can be numerical or classified outputs, depending on the specific application.

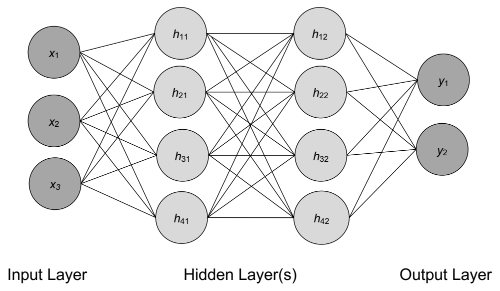


Figure 3 A simplified neural network structure. The input and output layers are connected through multiple hidden layers.

2.3 Confidential DNN

Many DNN-based applications (such as image processing, object detection, medical records, and financial transactions) deal with sensitive data and require protection against tampering or theft of intellectual property. When protecting model parameters is needed, an emerging approach is to execute critical DNN layers inside trusted enclaves. As enclaves have limited memory and DNN models are generally large, one common approach (used for general-purpose systems) is to execute the DNN workload layer-by-layer. This is known as “layer-based partitioning,” where each layer forms an independent partition. Each partition includes weights and biases and is stored in a separate encrypted file. The enclave stores the decryption key. The (encrypted) partition file is loaded into shared memory and decrypted by a trusted application on the secure side.

▶ **Note.** We do not intend to modify or improve the confidential DNN techniques – a large number of literature exists to show how DNN models can be protected using TrustZone or other TEEs [16,31,32,41]. Our focus in this paper is to make existing (non-time-aware) confidential DNN ideas used for general-purpose systems adaptable for time-critical applications (viz., a set of periodic tasks with deadline constraints controlled by a real-time scheduling policy).

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 $\Gamma = \{\tau_1, ...\tau_n\}$ 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 2.3), 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 $C_i^a = C_i^{dec} + C_i^{com}$ is the computation inside the enclave, where C_i^{dec} is the time required for decryption of the layers information and C_i^{com} is the computation time of task τ_i.

We assume the tasks follow the earliest deadline first (EDF) scheduling policy [40]. 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 OS [29] 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 Assumptions on Adversarial Capabilities

The pre-trained model (e.g., parameter and hyperparameter values such as structure or properties of the particular DNN) is deployed to the real-time platform before the system is operational. We consider an adversary seeks to gain access to sensitive information of the model. Our focus is on protecting the inference operations of the DNN model. Attackers may have access to the input data, but they will not obtain any information about the model architecture or the final inference as long as they execute within the enclave. The attackers may know all the task periods and their execution times. We further assume that the adversaries cannot bypass the TEE protection mechanisms. Note that similar assumptions are used by other researchers [32, 42].

Following the convention, we assume that the parameters of the pre-trained model are kept encrypted in the local storage. The hyperparameters of the model are kept unencrypted in the normal world as they are generally well-known to the public and do not disclose any sensitive information about the input or training data [23]. During inference (i.e., invocation of a job), the input data and model parameters are loaded into the enclave memory, and the required inference operation is carried out after the decryption of the model parameters.

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 [33]. In contrast, most DNN models need 100+ MB [38]. If a DNN model is too large, then the model may fail to execute inside the enclave. To illustrate this, we conducted experiments on Raspberry Pi running OP-TEE and Darknet [35]. For large models, Darknet was unable to load to model (Listing 1). Hence, we used a model compression technique using *Deep Compression* [19] to reduce the model size as we present below. Listing 2 shows the case after trimming that allowed us to load and run the model successfully.

Listing 1 OP-TEE Log: Failed Invocation of a Large Model on Raspberry Pi.

```
Prepare session with the TA
Begin darknet
...
...
# darknetp:TEEC Invoke_Command(CONV) failed
```

Listing 2 OP-TEE Log: Successful Inference.

```
Prepare session with the TA
Begin darknet
...
...
user CPU start:  0.029851; end: 0.029851
kernel CPU start:2.916167; end: 2.917143
Max: 2756  kilobytes
vmsize:545460850536; vmrss:365072222916;
   vmdata:545460847464; vmstk:132;
   vmexe:412; vmlib:545460848828

// Successful inference and job completion
```

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 [33]. 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 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).

Algorithm 1 Model Compression.

```
1: Input: $w_{ij}$ , $\lambda$
2: Output: Compressed Size ($w'_{ij}$)
3: Prune the network below a certain threshold $\lambda$ following state-of-the-art techniques [20].
4: Retrain the network.
5: Quantize the weights of model: $r = \frac{mb}{m \log_2 k + kb}$    ▷ Plugging the value of r from approximation
   percentage (i.e., $(1 - \theta_{ij}\%)$ ) 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 ($\mathcal{W}_i$), TEE Capacity $\delta$
2: Output: Resized model size set ($\mathcal{W}'_i$)
3: $\mathcal{W}'_i = [\,]$    ▷ Initialize to an n empty array
4: for $j = 1$ to $L_i$ in $\mathcal{W}_i$ do
5:     if $w_{ij} > \delta$ then
6:         Optimized the layer using Algorithm 1
7:         $\mathcal{W}_i{}' \leftarrow w_{ij}{}'$
8:     else
9:         $\mathcal{W}_i{}' \leftarrow w_{ij}$
10:    end if
11:    $j \leftarrow j + 1$
12: end for
13: return Resized model ($\mathcal{W}'_i$)
```

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. [20] 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 [33] 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 layers 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 [22,41]. 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. The following shows how we can adapt the EDF scheduler for a conventional layer-wise partitioning technique. We refer to this EDF variant as DeepTrust$^{\text{RT}}$-LW.

4.2 DeepTrust$^{\text{RT}}$-LW: Real-Time Layer-wise DNN Execution

Our first approach – DeepTrust$^{\text{RT}}$-LW– sequentially partitioned the layers. They are then transferred to the enclave and scheduled using EDF policy. This is feasible since there is no cross-dependency between any two layers, and each layer can be computed sequentially independently [22]. Traditional EDF schedulability conditions often involve checking many relative deadline points to assess the schedulability of a taskset up to the hyperperiod [30, 43]. 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 [43]. To determine the schedulability conditions for DeepTrust$^{\text{RT}}$-LW, we use the existing QPA-based EDF timing analysis technique [43] and adapt it to our DNN-based workload. We choose QPA-based analysis instead of others [30] 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 of DeepTrust$^{\text{RT}}$.

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 [15, 39]. 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 DeepTrust$^{\text{RT}}$-LW, 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 1 in Section 4.3 for a formal derivation. Note that, in DeepTrust$^{\text{RT}}$-LW, $n_i^{cs} = L_i$. 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 $T_{min} = \min\{T_1, T_2, \cdots, T_n\}$.

4.3 Workflow and Analysis of DeepTrust$^{\text{RT}}$-LW

As mentioned before, in DeepTrust$^{\text{RT}}$-LW, layers are sequentially executed inside the enclave, and the tasks are scheduled using the EDF policy. Algorithm 3 presents steps for the schedulability checks. 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).

Algorithm 3 DeepTrust$^{\text{RT}}$-LW Schedulability Checking.

```
1:  Input: Real-time taskset ($\Gamma$)
2:  Output: Taskset schedulability
3:  $t \leftarrow \max\{T_1, T_2, \cdots T_n\}$
4:  $T_{min} \leftarrow \min\{T_1, T_2, \cdots T_n\}$
5:  while $t > T_{\min}$ do
6:      $h(t) \leftarrow \sum_{i=1}^{n} \lfloor \frac{t}{T_i} \rfloor (C_i^a + L_i \times C^{cs})$        ▷ Calculate $h(t)$ for the given $t$
7:      if $h(t) + b(t) > T_{\min} \wedge h(t) + b(t) < t$ then
8:          $t \leftarrow h(t) + b(t)$
9:      else if $h(t) + b(t) \leq T_{\min}$ then
10:         Taskset is schedulable
11:         Break
12:     else
13:         Taskset is not schedulable
14:         Break
15:     end if
16: end while
```

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

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

Determining the Processor Demand Function. The following lemma shows the expression for $h(t)$.

▶ **Lemma 1.** *The maximum execution time required by the system contiguous interval of length t, 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 [43], $h(t) = \sum_{i=1}^{i=n} \max\{0, 1 + \lfloor \frac{t-D_i}{T_i} \rfloor\} \times C_i$. Replacing $D_i = T_i$ in the above equation (since we have an implicit deadline system) and after simplification, $h(t)$ 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: $h(t) = \sum_{i=1}^{i=n} \lfloor \frac{t}{T_i} \rfloor \times (C_i^a + n_i^{cs} \times C_i^{cs})$. ◀

Overhead Analysis. 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 DeepTrust$^{\text{RT}}$-LW, 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 DeepTrust$^{\text{RT}}$-LW 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 $\tau_1 > \delta$. DeepTrust$^{\text{RT}}$-LW 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.

4.4 The Need for an Efficient Scheduler

Despite DeepTrust$^{\mathrm{RT}}$-LW ensures real-time guarantees (for schedulable tasksets), as we shall see below (and also from our evaluation in Section 6), it results in poorer schedulability. This is because each switching results in extra SMC invocation, which increases task response times. For example, OP-TEE (an open-source TrustZone port for Linux) [29] performs five API calls to initiate and teardown a TEE session. 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 a simple 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 DeepTrust$^{\mathrm{RT}}$-LW 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 swich 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 DeepTrust$^{\mathrm{RT}}$-LW 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.

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.* We refer to this technique DeepTrust$^{\mathrm{RT}}$-FUSION. Figure 4 illustrates a high-level schematic for two tasks. In this case, DeepTrust$^{\mathrm{RT}}$-LW misses deadlines for Task 2 due to multiple context switch overheads. However, when we fuse the layers in DeepTrust$^{\mathrm{RT}}$-FUSION, we save context switch costs, thus allowing both tasks to meet deadlines.

We note that task fusion has been used in literature for TEE-enabled conventional (i.e., non-learning enabled) fixed-priority real-time systems to reduce TEE context switch overheads [34]. 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 with shorter deadlines so that *(a)* enclave utilization (capacity usage) is maximized, i.e., fit as many layers as possible, and *(b)* tasks do not miss deadlines. Our selection process, as described in Section 5, is inspired by the bin-packing heuristics (such as best-fit) [13] used in partitioned multiprocessor scheduling. We now discuss DeepTrust$^{\mathrm{RT}}$-FUSION scheduler in detail and derive schedulability conditions.

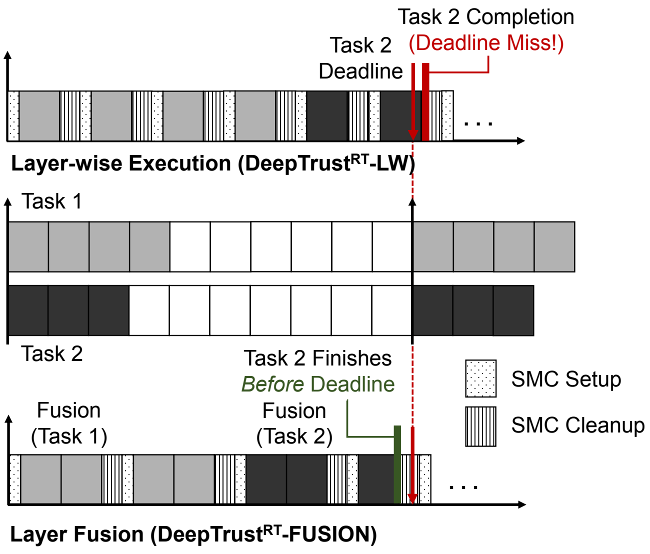


Figure 4 Key intuition of model fusion: when the tasks are executed layer-wise (DeepTrust^{RT}-LW, top schedule), Task 2 misses deadline due to multiple context switch overheads. However, in a multi-layer execution approach (DeepTrust^{RT}-FUSION, bottom schedule), multiple layers are fused together, which reduces context switch overheads and all the tasks meet their deadlines.

5 DeepTrust^{RT}-FUSION: Multi-layer Task Fusion

Fusing multiple layers from multiple tasks can significantly save context switch overheads compared to DeepTrust^{RT}-LW approach. For example, AlexNet-squeezed [18] has 16 layers. If the system follows DeepTrust^{RT}-LW, 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 fuse layers using DeepTrust^{RT}-FUSION, we can finish the execution with two context switches. We now illustrate how DeepTrust^{RT}-FUSION improves schedulability. Fusion works independent of task period types (i.e., harmonic/non-harmonic), as we illustrate with an 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

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 DeepTrust^{RT}-LW). Let us now consider the harmonic case. For 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.

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

5.1 Workflow of DeepTrust$^{\text{RT}}$-FUSION

DeepTrust$^{\text{RT}}$-FUSION aims to maximize the usage of TEE capacity. Hence, we send the maximum number of layers that TEE can support to reduce the SMC context switching overhead. 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 4 formally presents DeepTrust$^{\text{RT}}$-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, DeepTrust$^{\text{RT}}$-FUSION scheduler picks the fuse candidates (for instance, the loops in Line 8-Line 11, Algorithm 4). Let $\Omega(t)$ be the set of all tasks scheduled by using the vanilla EDF (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 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 $(\delta - \delta_1 = 1)$.

Algorithm 4 DeepTrust$^{\text{RT}}$-FUSION: Task Fusion and Scheduling.

```
1: Input: Real-time taskset ($\Gamma$), TEE-capacity $\delta$
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 using Lemma 3
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
```

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 schedulabe if $\forall t$, $U(t) < 1$ and $h(t) \leq t$. We now derive the expressions for $U(t)$ and $h(t)$ for DeepTrust$^{\text{RT}}$-FUSION.

▶ **Lemma 2.** *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*

$$U(t) = \sum_{i=1}^{i=n} \left(\frac{\lfloor \frac{t}{T_i} \rfloor \times C_i}{t} - \frac{n_i^s(t) \times C_i^{cs}}{t} \right). \tag{2}$$

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

In the following, we derive the processor demand function $h(t)$ for DeepTrust$^{\text{RT}}$-FUSION.

▶ **Lemma 3.** *The task set* Γ *is schedulable under DeepTrust*^{RT}*-FUSION if* $\forall t > 0$, $t < T_{max}$; $h(t) + b(t) \leq t$, $U(t) < 1$, *where*

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

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 $h(t) = \sum_{i=1}^{i=n} \lfloor \frac{t}{T_i} \rfloor C_i$. DeepTrust$^{\text{RT}}$-FUSION 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: $h(t) = \sum_{i=1}^{i=n} \left(\lfloor \frac{t}{T_i} \rfloor C_i - \frac{n_i^s(t) \times C_i^{cs}}{t} \right)$. ◀

We now calculate the performance benefits of DeepTrust$^{\text{RT}}$-FUSION, i.e., the reduction in SMC context switch counts when we use layer fusion.

▶ **Lemma 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:

$$n^s = \sum_{j=1}^{j=z} n_j^s = \sum_{j=1}^{j=z} (k_j - 1) C_j^{cs}. \tag{4}$$

◀

6 Evaluation

We evaluate our techniques on two fronts: *(a)* design-space exploration with various DNN workloads (Section 6.1) and *(b)* case study with a UAV autopilot system (Section 6.2). Our implementation is available on GitHub: `https://github.com/CPS2RL/DeepTrust-RT`.

Table 5 Systems & Workloads.

Parameters	**Description**
Hardware	4x ARM Cortex A53, 1 GB RAM (Raspberry Pi 3 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	$[50, 100]$ ms
Number of layers, L	$[5, 24]$
Weight, W	$[0.01, 7]$
Execution time inside TEE per layer, c_{ij}^a	$[0.1, 8]$ ms
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

6.1 Design-Space Exploration with Deep Learning Workloads

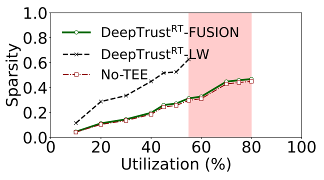
6.1.1 Simulation Setup

We evaluate the performance of our scheme using synthetically generated workloads, with parameters similar to that used in prior work [25]. We vary the system 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 100 ms. For deep learning workload, we used three popular DNN architectures: AlexNet-squeezed [21], YOLOv3-tiny [3], and Tiny Darknet [21]. 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 similar sizes of enclaves are used by OP-TEE. 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.

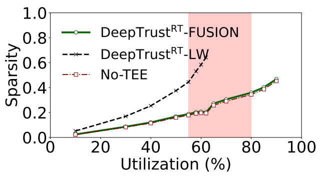
6.1.2 Schemes and Metrics

We compare DeepTrust$^{\mathrm{RT}}$-FUSION with layer-wise execution technique (DeepTrust$^{\mathrm{RT}}$-LW). For completeness, we also study a "non-secure" variant that does not consider any enclave. The schemes used in our evaluation are listed below.

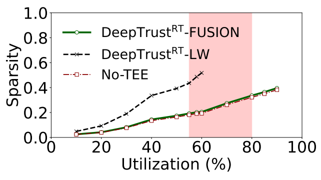
- **DeepTrust$^{\mathrm{RT}}$-LW**: Sends the layers *sequentially* (layer-wise) to the enclave (Section 4.2).
- **DeepTrust$^{\mathrm{RT}}$-FUSION**: Our proposed scheme that *fuses* multiple layers from multiple tasks (Section 5).
- **No-TEE**: DNN task execution *without* any enclave. The tasks follow EDF scheduling policy. In this case, model confidentiality is not enforced.



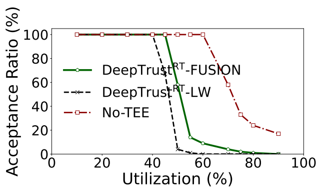
(a) Sparsity-vs-Util., n=5.



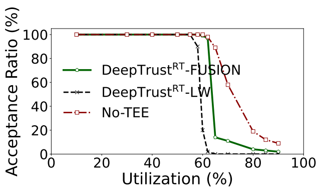
(b) Sparsity-vs-Util., n=10.



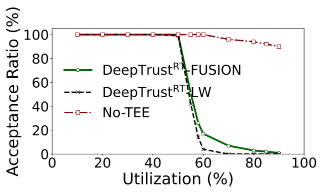
(c) Sparsity-vs-Util., n=15.



(d) Acc. Ratio-vs-Util., n=5.

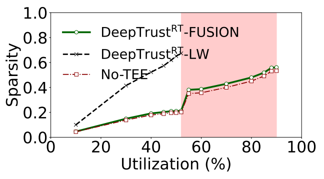


(e) Acc. Ratio-vs-Util., n=10.

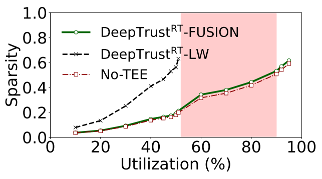


(f) Acc. Ratio-vs-Util., n=15.

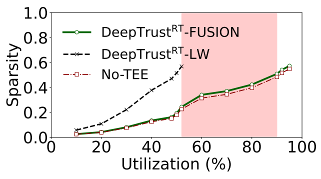
Figure 5 Sparsity and Acceptance Ratio with varying system utilization for $\{5, 10, 15\}$ tasks using AlexNet-squeezed [21] architecture. The red shaded regions show cases where DeepTrust$^{\text{RT}}$-LW cannot find schedulable tasksets while other schemes can. DeepTrust$^{\text{RT}}$-FUSION result in better schedulability compared to DeepTrust$^{\text{RT}}$-LW as the utilization increases with performance penalty (i.e., both Sparsity and Acceptance Ratio are close to the No-TEE case.



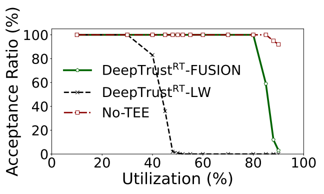
(a) Sparsity-vs-Util., n=5.



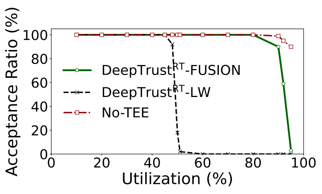
(b) Sparsity-vs-Util., n=10.



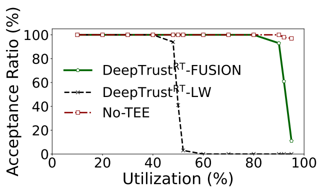
(c) Sparsity-vs-Util., n=15.



(d) Acc. Ratio-vs-Util., n=5.



(e) Acc. Ratio-vs-Util., n=10.



(f) Acc. Ratio-vs-Util., n=15.

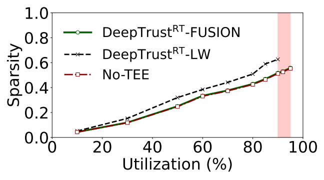
Figure 6 Sparsity and Acceptance Ratio using Tiny Darknet [21] architecture using a setup identical to that of Fig. 5. The findings are similar.

We tested the above schemes with the following two metrics.

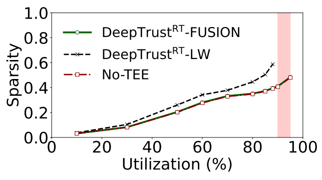
- **Sparsity**: Our newly introduced metric that shows the "spread" of the task (viz., the ratio between response time and period). Higher sparsity means tasks are completed late and that may result in poorer QoS in terms of the DNN inference process. A Sparsity value > 1 implies the task misses the 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

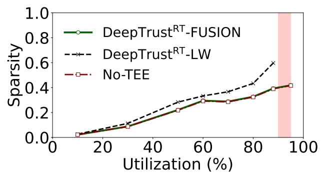
We first show the Sparsity and Acceptance Ratio for varying numbers of tasks ($n = 5$, $n = 10$, and $n = 15$) for the DNN workloads listed on Table 5. The x-axis of Fig. 5 shows the various taskset utilization for randomly generated taskset. The y-axis of Fig. 5a and Fig. 5d



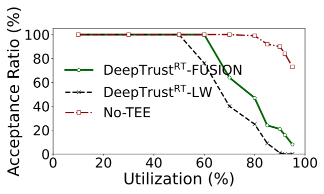
(a) Sparsity-vs-Util., n=5.



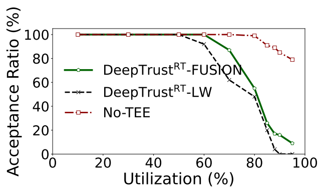
(b) Sparsity-vs-Util., n=10.



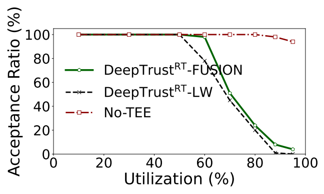
(c) Sparsity-vs-Util., n=15.



(d) Acc. Ratio-vs-Util., n=5.



(e) Acc. Ratio-vs-Util., n=10.



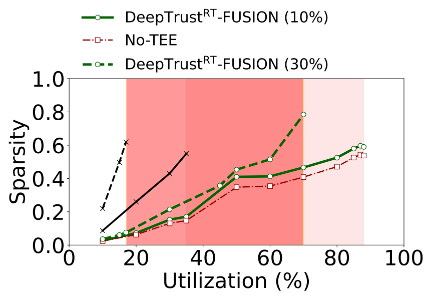
(f) Acc. Ratio-vs-Util., n=15.

Figure 7 Sparsity and Acceptance Ratio using YOLOv3-tiny [3] architecture using a setup identical to that of Fig. 5. Our findings are similar to Fig. 5 and Fig. 6.

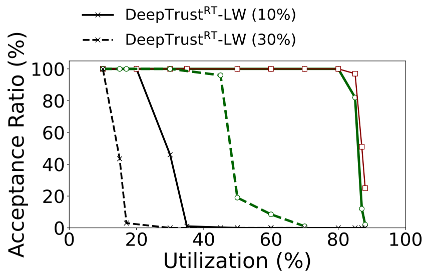
shows Sparsity and Acceptance Ratio, respectively. We show the Sparsity and Acceptance Ratio for DeepTrust^{RT}-FUSION (Green), DeepTrust^{RT}-LW (Black), and No-TEE (Black) schemes. The red shaded regions in the figure represent the cases where DeepTrust^{RT}-LW is unable to find any schedulable candidate while DeepTrust^{RT}-FUSION finds some. For lower utilization, all schemes show similar behavior. However, DeepTrust^{RT}-FUSION outperforms DeepTrust^{RT}-LW up to 3x as the utilization increases (i.e., DeepTrust^{RT}-LW is unable to find schedulable tasksets as the utilization reaches 60%). This is expected because layer-wise execution in DeepTrust^{RT}-LW 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 our scheme (both in terms of Sparsity and Acceptance Ratio) is close to No-TEE case (recall: No-TEE does not provide model confidentiality). Hence, DeepTrust^{RT}-FUSION can improve the security posture of the DNN tasks without significant overhead (close to the vanilla execution that does not have TEE support). In Figs. 6-7, we repeat the experiments with ImageNet1k datasets and obtain similar results. As the number of tasks increases (i.e., $n = 15$), the impact of context switching becomes more apparent, and hence, Acceptance Ratio in No-TEE case significantly outperforms DeepTrust^{RT}-LW and DeepTrust^{RT}-FUSION in highly utilized systems.

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. 8 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 larger context switch costs causes DeepTrust^{RT}-LW to perform poorly as delays accumulating by higher context switch duration lead to longer response times (higher Sparsity values), that in turn, cause more tasks to miss their deadlines (result in lower Acceptance Ratio).

DeepTrust^{RT}-FUSION outperforms DeepTrust^{RT}-LW, especially for high utilization scenarios. Further, the overhead of DeepTrust^{RT}-FUSION is negligible as its performance is close to the No-TEE case. Systems with longer TEE context switch delay can be significantly benefited by layer fusion compared to layer-wise partitioning.

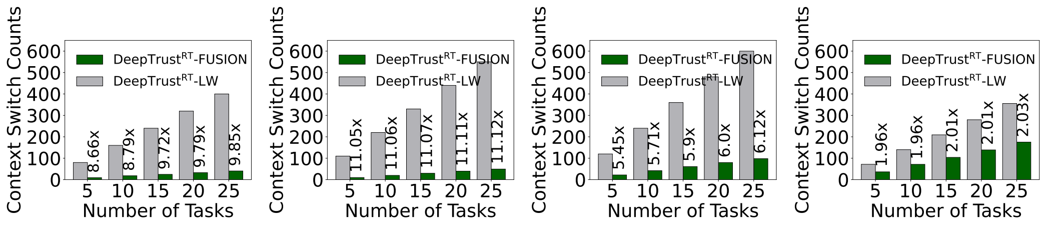


(a) Sparsity-vs-Util.



(b) Acc. Ratio-vs-Util.

Figure 8 Sparsity and Acceptance Ratio for a randomly generated workload with two different context switching overheads: *(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 DeepTrust$^{\mathrm{RT}}$-LW when compared to DeepTrust$^{\mathrm{RT}}$-FUSION, which in turn, reduces the percentage of schedulable tasksets. DeepTrust$^{\mathrm{RT}}$-FUSION performs identically to No-TEE for lower context switch delays.



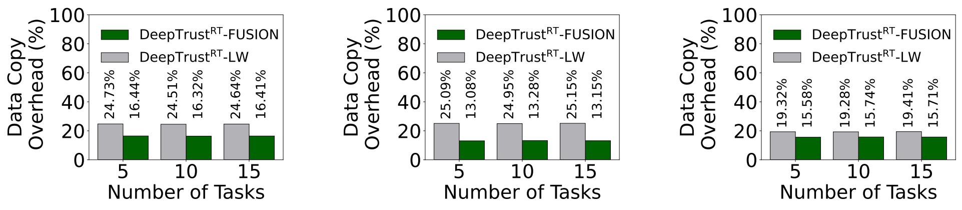
(a) AlexNet-squeezed. **(b)** Tiny Darknet. **(c)** YOLOv3-tiny. **(d)** Random workload.

Figure 9 Context switches overhead comparison for three architectures (e.g., Tiny Darknet, AlexNet-squeezed, YOLOv3-tiny) and one for a random taskset. DeepTrust$^{\mathrm{RT}}$-FUSION reduces context switch overhead 1.96x-11.12x compared to DeepTrust$^{\mathrm{RT}}$-LW.

In the next set of experiments (Fig. 9), we measure the number of SMC context switches for DeepTrust$^{\mathrm{RT}}$-FUSION and DeepTrust$^{\mathrm{RT}}$-LW. For this experiment, we set the system utilization 50%. Note that, as No-TEE does not have any enclave, there are no SMC calls (context switches). Hence, our plots exclude No-TEE in this case. As the figures show, DeepTrust$^{\mathrm{RT}}$-FUSION enables us to achieve a significant reduction in context switch counts compared to DeepTrust$^{\mathrm{RT}}$-LW (5.45x-11.1x) for all three architectures. This is because DeepTrust$^{\mathrm{RT}}$-FUSION groups multiple layers, hence reducing overall SMC calls.

DeepTrust^{RT}-FUSION can significantly reduce the number context switches (1.9x-11.1x) (see Fig. 9). This reduction of context switches also contribute to higher acceptance rate (see Figs. 5-7).

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. 10 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 $n = 15$) running on Raspberry Pi and OP-TEE. To calculate the end-to-end data copy overheads, we first measured the response times for No-TEE case and then subtracted these values from the response times of each of the DeepTrust$^{\mathrm{RT}}$ 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



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

Figure 10 DeepTrust$^{\text{RT}}$ 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 DeepTrust$^{\text{RT}}$-FUSION are less compared to DeepTrust$^{\text{RT}}$-LW due to fewer number of context switches.

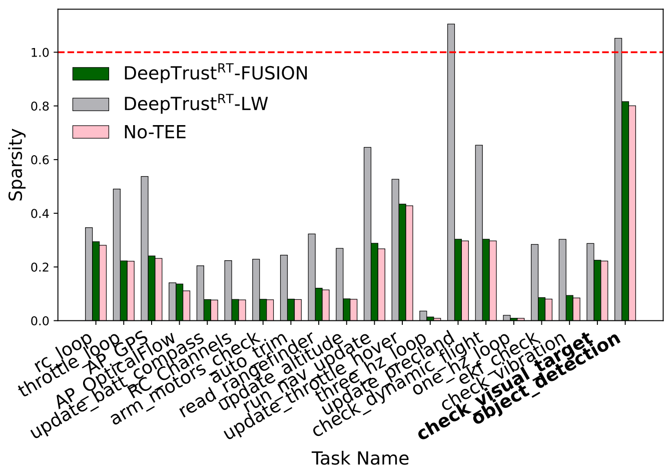


Figure 11 Sparsity for ArduPlilot controller tasks. Bold tasks are DNN inference workload. The red horizontal line denotes the deadline. The increasing number of context switches in DeepTrust$^{\text{RT}}$-LW caused a larger spread of tasks (higher Sparsity), and as a result, two tasks missed deadlines. Under DeepTrust$^{\text{RT}}$-FUSION, all tasks were able to meet deadlines.

samples and took the 90th 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 DeepTrust$^{\text{RT}}$-LW and DeepTrust$^{\text{RT}}$-FUSION are *(a)* 2.39 s and 1.34 s (AlexNet-squeezed), *(b)* 1.54 s and 2.95 s (Tiny Darknet), and *(c)* 5.62 s and 6.96 s (YOLOv3-tiny), respectively on Raspberry Pi+OP-TEE setup (recall: each SMC overheads could be as high as 20 ms, see Table 1). As the figure shows, the data copy overhead scales well with the increasing number of tasks (remains constant). Further, DeepTrust$^{\text{RT}}$-FUSION incurs lower overheads due to a reduced number of context switches, as we also observed in prior experiments (Fig. 9).

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. 11), we evaluate DeepTrust$^{\mathrm{RT}}$-FUSION, DeepTrust$^{\mathrm{RT}}$-LW, and No-TEE with a UAV autopilot system (ArduPlilot [1]) running on Raspberry Pi 3 [36]. 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. The periods of our DNN tasks were 5. The total system utilization (including two of the included DNN tasks) was 0.75. Each of the bars in Fig. 11 shows the various tasks and their Sparsity for each of the three schemes. The figure shows that due to high context switches, DeepTrust$^{\mathrm{RT}}$-LW misses deadlines for two real-time tasks (i.e., Sparsity > 1). In contrast, both DeepTrust$^{\mathrm{RT}}$-FUSION and No-TEE were able to meet all deadlines.

High SMC context switch overheads cause DeepTrust^{RT}-LW to miss deadlines for two real-time tasks. DeepTrust^{RT}-FUSION, in contrast, was able to meet all deadlines.

7 Discussion

In this work, we assume all layers execute inside TEEs. There exist use cases where not all layers have confidentiality requirements. For example, in image/voice recognition applications where the user may not want to reveal input and processed data, running initial input layers and final output layers inside TEE should be sufficient. Our future work will explore the variable number of TEE executions and analyze the performance trade-offs in a real-time context. Our research focuses on scheduling within a single enclave as existing TrustZone implementations support a single enclave. We will further investigate the feasibility and performance benefits of DeepTrust$^{\mathrm{RT}}$ running on multiple enclaves.

DeepTrust$^{\mathrm{RT}}$-FUSION 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 $\{l_j^{11}, \cdots, l_j^{13}\}$. If feasible (i.e., enclave has capacity), DeepTrust$^{\mathrm{RT}}$-FUSION fuses all seven layers. It could also be possible to obtain a "partial" slice of a layer in case a complete slice is 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. Sub-layer-based partitioning ideas will be explored in our future work.

We assume only inference tasks use TEEs. In practice, other (non-DNN) tasks could also use TEEs, thus potentially limiting enclave availability. DeepTrust$^{\mathrm{RT}}$-FUSION can be extended for such scenarios considering extra *slack* reclaimed from other non-inference tasks. The overall security of DeepTrust$^{\mathrm{RT}}$ relies on the underlying TEE architecture. However, TEEs could also be vulnerable, especially exposed to schedule-based attacks [4] for real-time context. One approach to limit such observability is to introduce "noise" in the scheduler [12]. For instance, instead of fusing the same set of tasks, DeepTrust$^{\mathrm{RT}}$-FUSION can be extended to select fusion candidates from different groups, thus limiting the predictability and hence reducing the chances of information leakage.

8 Related work

Research in confidential deep inference for real-time context is still in the early stages. In our preliminary (workshop) paper [8], we propose a layer-grouping idea for fixed priority systems. Unlike DeepTrust$^{\mathrm{RT}}$, our prior work does not provide formal timing guarantees.

AegisDNN [42] proposes to execute only a few layers that will be executed inside SGX-based TEEs. However, AegisDNN [42] is primarily designed for soft real-time systems and allows deadline misses. In contrast, DeepTrust^{RT} is designed for hard real-time systems.

There exists other work for general-purpose systems. DarkneTZ [32] 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 [17], Confidential DL [41], Occlumency [26]), for executing machine learning workloads inside TEEs. However, none of them consider real-time constraints.

SuperTEE [34] aims to reduce TEE task switching overhead. However, SuperTEE [34] is not designed for learning-enabled real-time systems and can not be directly adapted for DNN workloads. Researchers also propose techniques (e.g., Subflow [25], AppNet [10]) to make deep learning "time-aware," but they do not consider trusted execution aspects. The proposed research is one of the fundamental works that investigates time-aware confidential deep learning techniques for hard autonomous systems.

9 Conclusion

This research introduces techniques to enable real-time guarantees for confidential deep learning using trusted enclaves. We show how to slice a partition in a large deep-learning model to schedule using real-time schedulers such as EDF. We further propose an optimization using a novel idea of "layer fusion" that selectively groups multiple layers from various tasks to minimize TEE context switch overheads. By using the approach presented in this work, engineers of future autonomous systems will be able to design/schedule systems efficiently and measure overheads of deep neural inference workloads in a "quantifiable" way.

References

1 `https://github.com/ArduPilot/ardupilot`.

2 Intel software guard extensions: Intel SGX SDK for Linux OS. `https://intel.com`. accessed: 2020-06-30.

3 Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th international conference on advanced computing and communication systems (ICACCS)*, pages 687–694. IEEE, 2020.

4 Mohamed Anis Aguida and Monowar Hasan. 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)*, pages 301–304. IEEE, 2022.

5 Tiago Alves. TrustZone: Integrated hardware and software security. *Information Quarterly*, 3:18–24, 2004.

6 Mohammad Fakhruddin Babar and Monowar Hasan. CPS2RL/DeepTrust-RT. Software, NSF 2312006, WSU PG00021441 (visited on 06/06/2024). URL: `https://github.com/CPS2RL/DeepTrust-RT`.

7 Mohammad Fakhruddin Babar and Monowar Hasan. Real-time scheduling of TrustZone-enabled DNN workloads. In *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy*, pages 63–69, 2022.

8 Mohammad Fakhruddin Babar and Monowar Hasan. Real-time scheduling of Trustzone-enabled DNN workloads. In *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy*, pages 63–69, 2022.

9 Mohammad Fakhruddin Babar and Monowar Hasan. Trusted deep neural execution—a survey. *IEEE Access*, 2023.

10 Soroush Bateni and Cong Liu. ApNet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79. IEEE, 2018.

11 Hadjer Benkraouda and Klara Nahrstedt. Image reconstruction attacks on distributed machine learning models. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*, pages 29–35, 2021.

12 Chien-Ying Chen, Debopam Sanyal, and Sibin Mohan. Indistinguishability prevents scheduler side channels in real-time systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 666–684, 2021.

13 Jian-Jia Chen. Partitioned multiprocessor fixed-priority scheduling of sporadic real-time tasks. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 251–261. IEEE, 2016.

14 Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 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*, pages 1–23, 2019.

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

16 Tarek Elgamal and Klara Nahrstedt. Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 519–528. IEEE, 2020.

17 Akshay Gangal, Mengmei Ye, and Sheng Wei. HybridTEE: Secure mobile DNN execution using hybrid trusted execution environment. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2020.

18 Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-aware neural network design. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1638–1647, 2018.

19 Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint*, 2015. `arXiv:1510.00149`.

20 Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

21 Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint*, 2016. `arXiv:1602.07360`.

22 Md Shihabul Islam, Mahmoud Zamani, Chung Hwan Kim, Latifur Khan, and Kevin W. Hamlen. 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*, CODASPY'23, pages 153–164, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3577923.3583648`.

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

24 Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. On-device neural net inference with mobile GPUs. *arXiv preprint*, 2019. `arXiv:1907.01989`.

25 Seulki Lee and Shahriar Nirjon. 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)*, pages 15–29. IEEE, 2020.

26 Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019.

27 Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. TensorFI: A configurable fault injector for TensorFlow applications. In *IEEE International symposium on software reliability engineering workshops (ISSREW)*, pages 313–320. IEEE, 2018.

28 Ninghui Li, Wahbeh Qardaji, Dong Su, Yi Wu, and Weining Yang. Membership privacy: A unifying framework for privacy definitions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 889–900, 2013.

29 Linaro. Open portable trusted execution environment. `https://www.op-tee.org`, Accessed on 2021.

30 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

31 Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, and Mani Srivastava. 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*, pages 67–79, 2021.

32 Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

33 Alistair Moffat. Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4):1–35, 2019.

34 Anway Mukherjee, Tanmaya Mishra, Thidapat Chantem, Nathan Fisher, and Ryan Gerdes. Optimized trusted execution for hard real-time applications on COTS processors. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, pages 50–60, 2019.

35 Joseph Redmon. Darknet: Open source neural networks in C. `http://pjreddie.com/darknet/`, 2013–2016.

36 Matt Richardson and Shawn Wallace. *Getting started with Raspberry Pi*. O'Reilly Media, Inc., 2012.

37 Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

38 Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint*, 2014. `arXiv:1409.1556`.

39 Abhishek Singh and Sanjoy Baruah. Global EDF-based scheduling of multiple independent synchronous dataflow graphs. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 307–318, 2017. `doi:10.1109/RTSS.2017.00036`.

40 John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*, volume 460. Springer Science & Business Media, 1998.

41 Peter M VanNostrand, Ioannis Kyriazis, Michelle Cheng, Tian Guo, and Robert J Walls. Confidential deep learning: Executing proprietary models on untrusted devices. *arXiv preprint*, 2019. `arXiv:1908.10730`.

42 Yecheng Xiang, Yidi Wang, Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. AegisDNN: Dependable and timely execution of DNN tasks with SGX. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 68–81. IEEE, 2021.

43 Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58(9):1250–1258, 2009.