Real-Time Scheduling of TrustZone-enabled DNN Workloads

Mohammad Fakhruddin Babar
School of Computing, Wichita State University, USA
mxbabar@shockers.wichita.edu

Monowar Hasan
School of Computing, Wichita State University, USA
monowar.hasan@wichita.edu

ABSTRACT

Limited resources in embedded devices often hinder the execution of computation-heavy machine learning processes. Running deep neural network (DNN) workloads while preserving the integrity of the model parameters and without compromising temporal constraints of real-time applications, is a challenging problem. Although secure enclaves such as ARM TrustZone can ensure the integrity of applications, off-the-shelf implementations are often infeasible for DNN workloads — especially those with real-time requirements — due to additional resource and temporal constraints. This paper presents a real-time scheduling framework that enables the execution of resource-intensive DNN workloads inside TrustZone-enabled secure enclaves. Our approach reduces the resource overhead by fusing multiple layers of multiple tasks and running them all together inside the enclaves while retaining real-time grantees. We derive mathematical conditions that will allow the designer to test the feasibility of deploying DNN workload in a TrustZone-enabled system. Our comparisons with a standard fixed-priority real-time scheduler show that we can schedule up to 21.33% more tasksets in higher utilization (e.g., > 80%) scenarios.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**.

KEYWORDS

DNN, TrustZone, Real-Time Systems

ACM Reference Format:
Mohammad Fakhruddin Babar and Monowar Hasan. 2022. Real-Time Scheduling of TrustZone-enabled DNN Workloads. In *Proceedings of Proceedings of the 4th Workshop on CPS & IoT Security and Privacy (CPSIoTSec '22).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3560826.3563386

1 INTRODUCTION

Many cyber-physical applications (e.g., autonomous cars, unmanned aerial vehicles, smart robotics) often have "real-time" (i.e., stringent temporal) requirements for their correct operations. While traditionally, application tasks in real-time systems carry out more straightforward functionalities such as computations related to control loop updates, the advent of modern Internet-of-things (IoT)-specific applications and the emergence of edge computing require the end nodes to process large-scale data. For instance, modern real-time applications often require deep neural network (DNN)-based inferences for achieving intelligent features such as object recognition, image and video processing, and natural language processing [35]. Any late inference in the real-time decision-making process leads to detrimental behavior, which may damage the system, the environment, or the human users around it. The output of DNN algorithms is typically determined by input data, weight values, and intermediate results. Any manipulation of some (or all) of these parameters may lead to misclassification, as shown in recent attacks [28, 29, 31]. One way to protect safety-critical DNN tasks against such attacks is to protect the internals (i.e., DNN layers) from malicious data modifications. For instance, running the DNN inference tasks using secure enclaves (such as ARM TrustZone [25]) can ensure the integrity and confidentiality of the data and binary, even if the underlying OS is compromised.

Retrofitting TrustZone technology for securing real-time DNN workload is difficult since vanilla TrustZone and inference algorithms are not designed with the temporal and resource constraints of embedded cyber-physical devices. By design, secure enclaves keep the trusted computing base (TCB) and corresponding memory regions as small as possible to minimize the attack surface. In contrast, a typical DNN inference process contains millions of parameters and requires significant resources. For example, VGG-16 (an image classification task) [31] contains 138 million parameters that require 528 MB of memory for runtime computations. To put this into context, OP-TEE [25], an open-source TrustZone development stack, has only 8 MB of memory [2, 13], which makes it infeasible to execute a DNN inference task inside the enclave. Hence, there is a need to develop mechanisms that can execute resource-hungry inference tasks inside trusted enclaves without violating the temporal constraints of the system.

This paper introduces a technique to enable real-time aware DNN computations for TrustZone-enabled enclaves. The key idea of our approach is to *split the DNN tasks into multiple chunks* that can fit within the limited enclave capacity. However, splitting the inference tasks into chunks also increases overheads since the processor contexts need to switch back and forth from regular to trusted execution mode, and this may cause the tasks to miss their deadlines. Hence, we propose to *fuse multiple layers of DNN tasks and feed inside the enclave together.* This way, we can fit a larger, resource-intensive model inside a resource-constrained enclave and reduce the context switch overheads — thus making it suitable for real-time applications. This paper makes the following contributions:

The material in this paper is based upon work supported in part by the U.S. National Science Foundation (NSF) under grant NSF CNS 2152768. Any findings, opinions, recommendations, or conclusions expressed in the paper are those of the authors and do not necessarily reflect the views of sponsors.

CPSIoTSec '22, November 7, 2022, Los Angeles, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9876-3/22/11…$15.00
https://doi.org/10.1145/3560826.3563386

- A novel task fusion approach to execute resource-intensive DNN computations on TrustZone-enabled systems while retaining real-time guarantees; and
- A new scheduling framework and analytical model to derive the feasibility of deploying a given real-time DNN workload on TrustZone enclaves.

We compare our approach with the widely-used rate monotonic (RM) real-time scheduler [20] and find that fusing multiple layers of DNN inference tasks allows scheduling up to 21.33% more tasksets when compared to the vanilla system. We now start with background materials before we present the model and related assumptions.

2 OVERVIEW AND MOTIVATION

2.1 Background

2.1.1 Real-Time Systems. A system with real-time properties requires the application tasks to complete their executions before a predetermined *deadline*. This timing guarantee is crucial to avoid system failure. The system generally consists of a set of periodic tasks and follows a priority-driven scheduling policy. *Schedulability analysis* [26] formally verifies whether *all* tasks in the system meet their timing constraints (i.e., deadlines), and, if that is the case, the system is considered *safe*. Real-time systems are static, i.e., number of tasks, runtime, and activation patterns are predetermined at the design time and do not dynamically vary during system operation.

2.1.2 Trusted Execution and ARM TrustZone. Trusted execution environments (TEEs) allow a safe, isolated, and tamper-resistant runtime environment (TEE). ARM TrustZone [25] and Intel SGX [5] are the two most widely used TEE technologies. 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.

TrustZone is a hardware-based security feature for ARM devices. TrustZone is based on ARMv6, which includes a processor, memory, and peripheral security extensions. ARM TrustZone divides runtime operations into "normal" and "secure" worlds (see Fig. 1). In the normal world, a commodity OS (such as Linux) provides a traditional execution environment, while the secure world uses a small, trusted kernel (e.g., OP-TEE [13]). TrustZone-enabled systems with Cortex-A family processors can execute on four exception levels (EL0-EL3) and two modes (i.e., non-secure mode and secure mode). EL3 (monitor mode) runs a trusted firmware to switch between normal and secure modes. EL2 runs the hypervisor. EL1 hosts the kernel, and EL0 is used to execute the application code. The current execution state of the processor is determined by the non-secure (NS) bit. Two worlds can communicate using a shared memory region. A secure monitor call (SMC) is used to bridge the two worlds. When an SMC instruction is invoked from the normal world, the processor switches context from the normal world to the secure world (via monitor mode) and freezes normal world operation. TrustZone has dedicated memory sections for both worlds. The normal world cannot access secure memory, but the secure world can access normal memory.

2.1.3 Neural Network Inference. Neural network algorithms build a model based on training data in order to generate predictions without needing to be explicitly programmed. Figure 2 presents a simplified architecture of a neural network algorithm. During the inference process, the data from the inputs are sent through the *layers* where each layer performs matrix multiplications on the data. Final layer outputs might be either a number or a classification output, depending on the application. The most widely used neural network algorithms used in embedded and edge devices are artificial neural networks (ANN), deep neural networks (DNN), and convolutional neural networks (CNN).

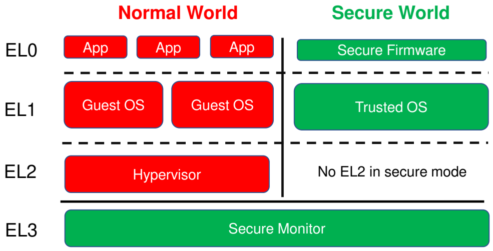

Figure 1: TrustZone Architecture for Cortex-A processors.

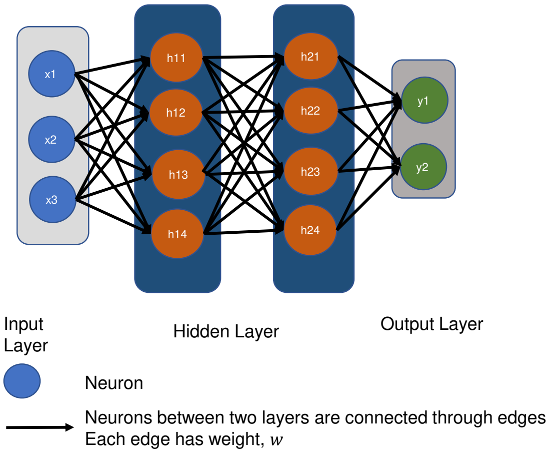

Figure 2: A Simplified Neural Network Structure.

ANN consists of an input layer, one or more hidden layers, and an output layer. Each node is connected to the others through an edge. Each edge has a weight and threshold linked with it. If a node's output exceeds a specific threshold value, the node is activated, and data is sent to the next layer of the network. A DNN is composed of numerous layers between the input and output that provides high accuracy [11, 12]. DNNs are typically feed-forward networks, where data flows from the input layer to the output layer without looping back. There are four types of layers that make up a DNN: *(i)* convolutional layers, *(ii)* activation layers, *(iii)* pooling layers, and *(iv)* fully-connected layers. Due to the computational complexity of convolution processes, convolutional layers are the most computationally intensive of the four types of layers. Using matrix multiplications in conjunction with convolutional filter operations is a particular case of DNNs, which are referred to as convolutional neural networks (CNNs). CNNs are used for image and video analysis when there are a large number of input variables to be processed. In this work, we focus on DNNs as this is the most general and widely used neural network algorithm.

2.2 Requirements for a Resource-aware DNN Inference Mechanism

Let us consider a pre-trained machine learning model used for performing inference for a given input (e.g., image). When enough resources (e.g., computational power and memory) are available and model security is not an issue, we can preload the weights and biases of each neuron into the memory to calculate neuron activations. However, such vanilla execution does not ensure the integrity of the model parameters. To provide data integrity, an alternative approach could be to allow the execution of the model layers inside the TrustZone enclave. However, preloading all these values (e.g., weights) requires a substantial amount of memory and may not be suitable for TrustZone-enabled systems. For instance, in OP-TEE [13] (an open-source software stack for TrustZone), the physical memory dedicated to the secure world is statically configured at build time. In the current version of OP-TEE, trusted applications are limited to 8 MB of secure world memory, thus preventing the execution of larger machine learning models. To further investigate this issue, we conducted experiments on OP-TEE and DarkneTZ [6] which is a TrustZone-based extension of Darknet [21] DNN models. When we ran the whole inference algorithm that includes 10 layers on the enclave, the process was terminated due to resource unavailability (see Listing 1, Line 5, marked red). In contrast, our scheme, which splits the whole inference process into layer-wise chunks was able to successfully perform the inference as shown in Listing 2 (Line 9, marked green).

```
1 Prepare session with the TA
2 Begin darknet
3 ...
4 ...
5 # darknetp:TEEC Invoke_Command(forward) failed 0xffff000c origin 0x4
```

Listing 1: OP-TEE Log: Failed Invocation.

```
1 Prepare session with the TA
2 Begin darknet
3 ...
4 ...
5 user CPU start:  4.663724; end: 22.591004
6 kernel CPU start:10.816184; end: 10.816184
7 Max: 16524  kilobytes
8 vmsize:971588; vmrss:16524; vmdata:53704; vmstk:132;
      vmexe:412; vmlib:2236
9 Loaded: 0.003270 seconds
```

Listing 2: OP-TEE Log: Successful Inference.

3 MODEL AND ASSUMPTIONS

3.1 System Model

We consider a uniprocessor real-time system running on a TrustZone-enabled platform. The system is consists of n real-time tasks $\Gamma = \{\tau_1, \ldots \tau_n\}$. Each task τ_i is characterized by $\tau_i = \{C_i^a, T_i, L_i, W_i\}$ where C_i^a is the worst case execution time of the task inside TEE, T_i is the inter-arrival time (period), L_i is the number of layers of the DNN model, and W_i is the size of the model. We assume the tasks follow the implicit deadline model, i.e., the tasks must complete before their next periodic arrival. We further assume that Γ is "schedulable" by a fixed-priority, preemptive real-time scheduler, i.e., the response time of each task (the time between arrival to completion) is less than its deadline.

Table 1: API Calls Required for Invoking a TEE Call.

API	Function
`TEEC_InitializeContext()`	Initialize connection
`TEEC_OpenSession()`	Open a new TEE session
`TEEC_InvokeCommand()`	Invokes a Command
`TEEC_CloseSession()`	Close the session
`TEEC_FinalizeContext()`	Close connection

We assume that the trusted enclave has a finite capacity δ, i.e., it can execute $M \geq 1$ layers together as long as the total resource requirements of those layers are less than δ. We express the worst-case execution time of τ_i as $C_i = C_i^{ns} + C_i^s$, where C_i^{ns} is the non-secure computation time and C_i^s is the worst case execution time inside TEE. Invoking a TEE session involves a series of API calls. For instance, OP-TEE requires 5 API calls (see Table 1) for instantiating and terminating a TEE session. Hence we represent C_i^s as $C_i^s = C_s^{st} + C_i^a + C_s^d$, where C_s^{st} is SMC set up time, C_i^a is actual (inference) computation of the task inside TEE and C_s^d is the SMC destroyed time. The tasks are scheduled by using the rate monotonic (RM) scheduling policy [20].

3.2 Threat Model

We assume that an adversary can access the non-secure components of the system. Our focus here is on protecting the inference operations of the DNN model. The attacker may have access to the input data, but they will not get any information about the model architecture or the final inference (since they are running inside the TEE). We further assume that the attacker may know all the task periods and their execution times. We do not consider any physical or hardware attack and assume that the adversary cannot bypass the TEE protection mechanisms.

4 MULTI-LAYER TASK MODEL

If a DNN model is too large to run entirely within the secure world, then the model may fail to execute, as shown in Listing 1. This limits each layer's size to the available memory capacity. This may require a complete redesign to reduce the number of neurons in each layer, potentially impacting model accuracy. In such cases, a better alternative could be to split the DNN model into smaller parts. This partitioning method is beneficial as the only values needed at a given time are the activation of the previous layer and the weights and biases for the current layer. Each layer contains its weight logs containing all the weights and biases for the neurons associated with that partition. For example, for two fully connected layers of m neurons, m activations, $m \times m$ weights, and m biases would be needed at a given time. This effectively limits the instantaneous memory footprint to that of a single layer, with the largest layer in the model determining the minimum amount of secure world memory needed. However, this approach necessitates context switching from the real world to the secure world, and this overhead is not negligible. If we perform layer-wise operations, we need to account for the context switching overheads for each layer execution. As we shall see, it is feasible to send multiple

Table 2: Example Taskset 1

Task	C^{ns}	C_s^{st}	C_i^a	C_s^d	C	T
τ_1	4	2	13	1	20	50
τ_2	4	2	5	1	12	100
τ_3	4	2	13	1	20	40

layers to reduce this context switching overhead. Based on this idea, we develop a *multi-layer task scheduling model* for secure DNN execution inside TrustZone TEEs.

We first start with an example taskset to show how fusing multiple secure tasks in the TEE can reduce the timing overhead (and, hence, can increase resource availability). As Table 2 shows, there are three tasks τ_1, τ_2, τ_3 where C_1, C_2, C_3 are respectively 20, 12, and 20 units and T_1, T_2, T_3 are 50, 100, and 40 units of time, respectively. In this taskset, $\sum C_i/T_i = 1.02 > 1$ and hence the taskset is not schedulable (since the system utilization is over 100%). In contrast, if we fuse τ_1 and τ_3, we can save 3 unit of time from $C_s^{st} + C_s^d = 3$, then $\sum C_i/T_i = 0.96 < 1$, then we can schedule this taskset as utilization is less than 100%.

4.1 Schedulability Conditions

We do offline (i.e., design-time) profiling for a given taskset and find the corresponding task scheduling. Our idea is to send the maximum number of layers that can be supported by TEE to reduce the SMC context switching overhead. For a given DNN task τ_i, 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 layers in task τ_i, then size of each layer will be $w_{i1}, w_{i2}, \ldots\ldots, w_{im}$, where $\sum_{j=1}^{i=L} w_{ij} = W_i$. We first check if the following condition holds: $(w_{i1} + w_{i2}) < \delta$. In this case, we check the following layers. We find the maximum value of k where $\sum_{j=1}^{j=k} w_{ij} = \delta_1 < \delta$, $\sum_{j=1}^{j=k+1} w_{ij} > \delta$. We find that transition point using this condition. If there is some extra capacity left (i.e., $\delta - \delta_1$), then we check the subsequent task to fit within this extra space. We find the maximum value of k for the subsequent task where $\sum_{j=1}^{j=k} w_{(i+1)j} < (\delta - \delta_1)$, $\sum_{j=1}^{j=k+1} w_{(i+1j} > (\delta - \delta_1)$. We check all invocation of the tasks until we reach the hyperperiod.[1] Once we obtain the schedule profile for one hyperperiod, we repeat that same pattern for all subsequent arrivals.

Let us assume we have two tasks τ_1, τ_2 each having 5 layers and $\delta = 5$. The size of τ_1 is 10 and the size of τ_2 is 5 units. WCET of τ_1 is 40 and WCET of τ_2 is 21 units. We cannot execute all the layers of τ_1 inside the enclave as size of $\tau_1 > \delta$. If we execute layer by layer, we need five SMC switching from the normal world for five layers for both tasks τ_1 and τ_2. If we send multiple layers of τ_1 that can be supported by TEE, it still requires three SMC switching i.e., $\{(w_{11}, w_{12}), (w_{13}, w_{14}), w_{15}\}$. For task τ_2, we need one SMC switching $(w_{21}, w_{22}, w_{23}, w_{24}, w_{25})$ as the size of $\tau_2 \leq \delta$. We need ten SMC switches for layer-by-layer operations to execute these two tasks. In contrast, it is possible to perform same objective by using only four SMC switches if we can send it by multiple layers. Note that if we send multiple layers of τ_1, we have still 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, $w_{11} + w_{12} + w_{21} = 5 \leq \delta$. Hence, we can fuse the first two layers from τ_1 and the first layer from τ_2, 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_{21}), (w_{13}, w_{14}, w_{22}), $(w_{15}, w_{23}, w_{24}, w_{25})$, i.e., we only need three SMC switches. The above example shows how to find the candidates for our proposed multi-layer fused task model (see Section 4.2).

4.1.1 Feasibility Analysis. If we can fuse n tasks from the Γ, then fused task τ^{com} is defined as (C^{com}, n^{com}). If we can fuse k tasks, then C^{com} can be measured using the following equation:

$$C^{com} = C_1 + \{C^s + C^d\} * n^{com} + \sum_{i=1}^{i=k} C_i^t, \tag{1}$$

where T_{hyp} is the hyperperiod of all the tasks. Here, n^{com} is the minimum SMC switching required for the fused task. If we can fuse two tasks τ_1, τ_2, we can say that $n^{com} \leq (n_a^{worst} + n_b^{worst})$. If there exists k number of fused tasks within a hyperperiod, we calculate n^s, where n^s is the total SMC saved due to fused task in a hyperperiod where $n^s = \sum n^{worst} - n^{com}$. For a given taskset, we derive the utilization using the following equation:

$$U_T = \sum_{i=1}^{i=n} \frac{\frac{T_{hyp}}{T_i} * C_i}{T_{hyp}} - \frac{n^s * (C^s + C^d)}{T_{hyp}}. \tag{2}$$

A task must start after the job's arrival and end before the job's period to guarantee schedulability. If candidates of fused tasks are $\tau_i, \tau_j, \ldots., \tau_k$, then all tasks will meet the individual deadline if they hold the following condition:

$$T_{high} - C^{com} - \sum C_p > 0. \tag{3}$$

In the above equation, C_p is the summation of the worst-case execution of all the higher priority tasks than τ_i that needs to be executed, and T_{high} is the period of the highest priority task in the fused group.

4.2 Algorithm

Algorithm 1 formally presents our idea. Let Ω is the set of all task scheduled by using the RM algorithm. For a given taskset we find the set of layers S for each SMC switching to send to TEE (Line 11). For each task, we find the transition point k and include layers l to k in set S. Next, we calculate the corresponding candidate by following the condition in Line 16. We check the schedulability following equation 3 (Line 25). If the task is schedulable and the size of the layers is less than δ (Line 24), we include those layers in set S and remove those layers from Ω (Line 17). When we finish checking one task, we check all other remaining tasks to see whether we can fit more layers of information in the set S. We then move on to find the next candidate.

4.3 Examples and Illustrations

Example #1 (Meeting temporal constraints). Table 3 and Table 4 presents 3 tasks and their corresponding parameters. Each task indicates a DNN model. Each task's size is shown in column W, and its layer-wise size is shown in Table 4. Specifically, n^l is the number of SMC switching if it executes layer-wise, and n^{worst} is

[1] A hyperperiod is the least common multiple (LCM) of the periods of the tasks, i.e., hyperperiod $T_{hyp} = LCM(T_1, T_2, \cdots, T_n), \forall \tau_i \in \Gamma$.

Algorithm 1 Fused Task Scheduling

```
1:  Input: Real-time taskset (Γ), TEE-capacity $\delta$
2:  Output: Taskset schedulability
3:  INITIALIZATION
4:  $\Omega = RM\_Scheduling(\Gamma)$                ▷ Obtain RM scheduling
5:  $L = \{L_1, L_2, \cdots\}$            ▷ L is the set of all DNN tasks layer
6:  $W = \{W_1, W_2, \cdots\}$                   ▷ Size of each DNN task
7:  $W_i = \{w_{i1}, w_{i2}, \cdots, w_{il}\}$         ▷ Size of each layer
8:  BEGIN                                  ▷ Find layers to send to TEE
9:  while TRUE do
10:     S=FIND_LAYERS_TO_SEND{Ω}          ▷ See Line 21 for definition
11:     Send S to TEE
12: end while
13: END
14: function FIND_TRANSITION_OF_LAYERS(Ω)
15:     if $\sum_{j=m}^{j=k} w_{ij} = \delta_1 < \delta$ and $\sum_{j=1}^{j=k+1} w_{ij} > \delta$ then
16:         Remove $w_{im}, \cdots, w_{ik}$ from Ω
17:     end if
18:     return $w_{im}, \cdots, w_{ik}$
19: end function
20: function FIND_LAYERS_TO_SEND(Ω)
21:     while $\Omega \neq NULL$ do
22:         S=FIND_TRANSITION_OF_LAYERS(Ω)          ▷ See Line 15
23:         if $sum(S) \leq \delta$ then
24:             Check schedulability condition using Eq. (3)
25:             if Schedulable then
26:                 Check next task
27:             else
28:                 break
29:             end if
30:         else
31:             break
32:         end if
33:     end while
34:     return S
35: end function
```

Table 3: Example Taskset 2

Task	T	L	W	C_s^a	$C_s^{st} + C_s^d$	n^{worst}	C
τ_1	60	5	10	31	3	3	40
τ_2	120	5	5	18	3	1	21
τ_3	120	5	5	18	3	1	21

Table 4: Layer-wise Parameters for Taskset 2

Task	w_1	w_2	w_3	w_4	w_5	n^{worst}	n^l
τ_1	2	2	2	2	2	3	5
τ_2	1	1	1	1	1	1	5
τ_3	1	1	1	1	1	1	5

the worst case SMC switching if we send the multiple layers of the same task that can be sustained by TEE.

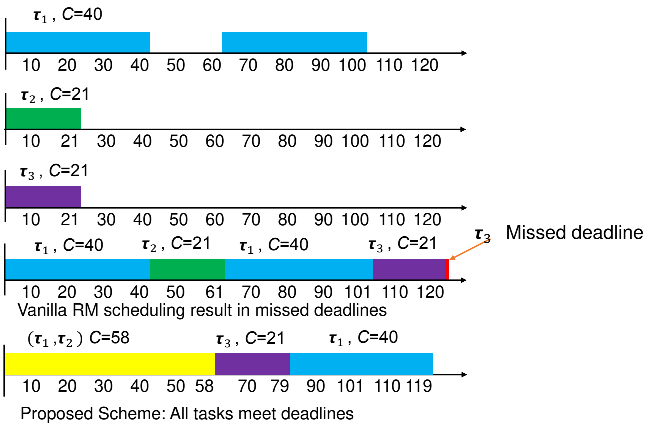

Figure 3: Schedule for Taskset 2.

Figure 3 depicts the corresponding task arrival and their offline profiled scheduling. We can see that the fused task is (τ_1, τ_2) and needs 3 SMC switches. If we execute all the tasks within the hyperperiod, layer-wise, we need 20 SMC switches. If we follow our multi-layer switching, we need only 7 SMC switches. We send all the layers of the task-set 3 within its hyperperiod in the following manner: (w_{11}, w_{12}, w_{21}), (w_{13}, w_{14}, w_{22}), $(w_{15}, w_{23}, w_{24}, w_{25})$, $(w_{31}, w_{32}, \cdots, w_{35})$, (w_{11}, w_{12}), (w_{13}, w_{14}), (w_{15}). As we can see from Fig. 3, when we follow vanilla RM scheduling τ_3 misses its deadline (pointed by the red arrow). In contrast, tasks scheduled by using our scheme can meet *all* deadlines (see Fig. 3).

Example #2 (Reduction of context switch overheads). Table 5 presents 4 tasks in Taskset #3. Each task’s size is listed in column W and the layer-wise parameters are shown in table 4. In Fig. 4, we can see the corresponding task arrival and scheduling inside TEE. If we execute all the tasks within the hyperperiod layer-wise, we need 70 SMC switches. In contrast, if we follow our multi-layer switching, we need only 16 SMC switches.

Table 5: Example Taskset 3

Task	T	L	W	C_s^a	$C_s^{st} + C_s^d$	n^{worst}	C
τ_1	60	5	5	10	2	1	12
τ_2	80	5	10	23	2	3	29
τ_3	240	5	10	23	2	3	29
τ_4	120	5	5	10	2	1	12

5 EVALUATION

5.1 Simulation Setup

We evaluate the performance of our scheme using synthetically generated workloads. We used parameters similar to that used in prior work [18]. We varied the system utilization from 0% to 80% with an increment of 10 and from 80% to 100% with an increment of 2. For different system utilization $u \in [0, 10, \cdots, 100]$%, we have generated $N_u = 100$ tasksets. Each taskset has [2, 10] tasks. The task periods are randomly selected from [50, 100] ms. We assume that the enclave capacity $\delta = 6$ MB and SMC context switching overhead $c_s^{st} + c_s^d$ is 3 ms. Table 6 lists the key simulation parameters. Our implementation is publicly available [34].

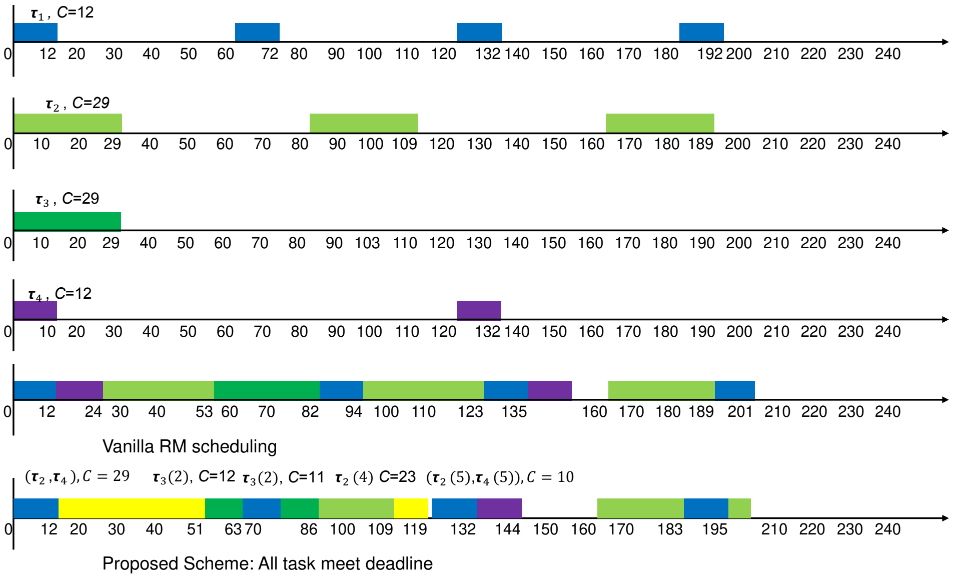

Figure 4: Schedule for Taskset 3.

Table 6: Simulation Parameters

Parameters	Value
Enclave capacity, δ	6 MB
Utilization, U	0%-100%
Period T	[50, 100] ms
Number of layers, L	[2, 20]
Weight, W	[2, 20]
Execution time inside TEE, c_s^a	[10, 50] ms
SMC overhead, $c_s^{st} + c_s^d$	3 ms
Number of tasks, n	[2, 10]
Number of taskset for each utilization, N_u	100

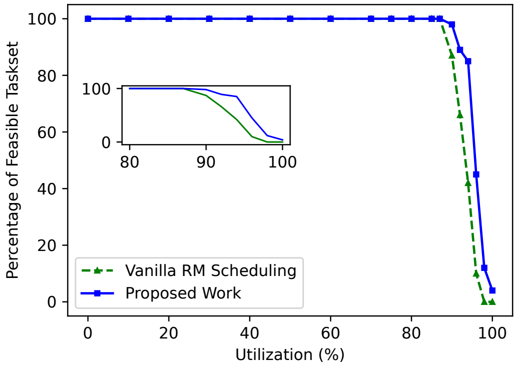

Figure 5: Percentage of task-set for schedulable by RM scheduling and our proposed scheme. Task fusion (solid line) results in better schedulability when the system utilization increases.

5.2 Results

We compare our scheme with vanilla RM scheduling policy [20]. The x-axis of Fig. 5 shows the percentage of feasible tasks (i.e., ratio of the number of tasksets that meets timing constraints over the total generated one), and the y-axis shows the system utilization for both RM (dotted line) and proposed scheme (solid line). For lower utilization, this difference in response times is insignificant; hence, both schemes show identical behavior. However, our proposed scheme outperforms RM scheduling for higher utilization ($> 85\%$). This is because when multiple tasks are fused, it reduces context switch (i.e., SMC) overheads and result in shorter response times. As a result, more tasksets find schedulable (i.e., tasks completed before deadlines). From our experiments, we find that, on average, our scheme finds 21.33% more schedulable tasksets compared to the vanilla RM scheduling.

6 RELATED WORK & CONCLUSION

Researchers propose various strategies (e.g., HybridTEE [8], Confidential DL [7], DarkneTZ [6], Occlumency [33]) for executing ML (i.e., DNN) workloads inside TEEs. However, none of them consider real-time constraints. DarkneTZ [6] and AegisDNN [32] propose 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. Perhaps the closest line of work to ours is SuperTEE [18] which aims to reduce task switching overhead. However, SuperTEE does not consider scheduling machine learning workloads inside limited TEE capacity. In this paper, we propose a resource-aware multi-layer task scheduling model that focused on optimizing TEE resources. By fusing multiple layers together our scheme results in shorter response times, and hence, better schedulability when compared to a traditional RM scheduling policy. To the best of our knowledge, this is one of the first attempts to enable real-time scheduling of DNN workloads on TrustZone-enabled systems.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25 (2012), 1097–1105.

[2] Linaro. Accessed on 2021. Open portable trusted execution environment. https://www.op-tee.org

[3] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) MobiLeNets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:170404861

[4] "GlobalPlatform", TEE system architecture, 2011, [online] Available: http://www.globalplatform.org/specificationsdevice.asp.

[5] Intel Software Guard Extensions (Intel SGX) SDK for Linux OS. http://intel.com. Accessed: 2020-06-30.

[6] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "DarkneTZ: Towards model privacy at the edge using trusted execution environments," in International Conference on Mobile Systems, Applications, and Services (MobiSys), 2020, p. 161–174.

[7] P. M. VanNostrand, I. Kyriazis, M. Cheng, T. Guo, and R. J. Walls, "Confidential deep learning: Executing proprietary models on untrusted devices," in arXiv:1908.10730, 2019.

[8] A. Gangal, M. Ye and S. Wei, "HybridTEE: Secure Mobile DNN Execution Using Hybrid Trusted Execution Environment," 2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2020, pp. 1-6, doi: 10.1109/AsianHOST51057.2020.9358260.

[9] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," 2015 IEEE Trustcom/BigDataSE/ISPA, 2015, pp. 57-64, doi: 10.1109/Trustcom.2015.357.

[10] Intel Software Guard Extensions (Intel SGX) SDK for Linux OS. http://intel.com. Accessed: 2020-06-30.

[11] Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E (2015) Deep learning applications and challenges in big data analytics. J Big Data 2(1):1

[12] Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning, vol 1. MIT Press, Cambridge

[13] "OP-TEE (Open Portable Trusted Execution Environment)." https://www.op-tee.org/. Accessed: 2018-05-27.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788

[15] Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105.

[16] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark Analysis of Representative Deep Neural Network Architectures. IEEE Access 6 (2018), 64270–64277.

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

[18] Anway Mukherjee, Tanmaya Mishra, Thidapat Chantem, Nathan Fisher, and Ryan Gerdes. 2019. Optimized trusted execution for hard real-time applications on COTS processors. In Proceedings of the 27th International Conference on Real-Time Networks and Systems (RTNS '19). Association for Computing Machinery, New York, NY, USA, 50–60. https://doi.org/10.1145/3356401.3356419

[19] R. Liu and M. Srivastava, "Protc: Protecting drone's peripherals through arm trustzone," in Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, pp. 1–6, ACM, 2017.

[20] Lehoczky, John, Lui Sha, and Yuqin Ding. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior." RTSS. Vol. 89. 1989.

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

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015.Going deeper with convolutions. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 1–9.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 770–778.

[24] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger.2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 4700–4708.

[25] T. Alves and D. Felton. 2004. TrustZone: Integrated hardware and software security. Tech. In-Depth 3, 4 (2004), 18–24.

[26] Sorin Manolache, Petru Eles, and Zebo Peng. 2004. Schedulability analysis of applications with stochastic task execution times. ACM Trans. Embed. Comput. Syst. 3, 4 (November 2004), 706–735. https://doi.org/10.1145/1027794.1027797

[27] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM (JACM), vol. 20, no. 1, pp. 46–61, 1973.

[28] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben. BinFI: an efficient fault injector for safety-critical machine learning systems. In International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2019.

[29] G. Li, K. Pattabiraman, and N. DeBardeleben. Tensorfi: A configurable fault injector for tensorflow applications. In IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2018.

[30] A. S. Rakin, Z. He, and D. Fan. Bit-flip attack: Crushing neural network with progressive bit search. In IEEE/CVF International Conference on Computer Vision, 2019.

[31] Benali Amjoud, Ayoub, and Mustapha Amrouch. "Convolutional Neural Networks Backbones for Object Detection." Image and Signal Processing: 9th International Conference, ICISP 2020, Marrakesh, Morocco, June 4–6, 2020, Proceedings vol. 12119 282–289. 5 Jun. 2020,

[32] Y. Xiang, Y. Wang, H. Choi, M. Karimi and H. Kim, "AegisDNN: Dependable and Timely Execution of DNN Tasks with SGX," 2021 IEEE Real-Time Systems Symposium (RTSS), 2021, pp. 68-81, doi: 10.1109/RTSS52674.2021.00018.

[33] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19). Association for Computing Machinery, New York, NY, USA, Article 46, 1–17. https://doi.org/10.1145/3300061.3345447

[34] Multi-layer task fusion model implementation, https://github.com/CPS2RL/Scheduling-of-TrustZone-enabled-DNN-Workloads

[35] S. Lee and S. Nirjon, "SubFlow: A Dynamic Induced-Subgraph Strategy Toward Real-Time DNN Inference and Training," 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2020, pp. 15-29, doi: 10.1109/RTAS48715.2020.00-20.