RT-MDM: Real-Time Scheduling Framework for Multi-DNN on MCU Using External Memory

Sukmin Kang[*]
Dept. of Semiconductor and Display Engineering, Sungkyunkwan University, South Korea
Samsung Electronics Co., Ltd., South Korea
sukmin.kang@skku.edu

Seongtae Lee[*]
Dept. of Computer Science and Engineering, Sungkyunkwan University, South Korea
yuns0509@skku.edu

Hyunwoo Koo
School of Electronic and Electrical Engineering, Sungkyunkwan University, South Korea
koowoo3@skku.edu

Hoon Sung Chwa
Dept. of Electrical Engineering and Computer Science, DGIST, South Korea
chwahs@dgist.ac.kr

Jinkyu Lee[†]
Dept. of Computer Science and Engineering, Sungkyunkwan University, South Korea
jinkyu.lee@skku.edu

ABSTRACT

As the application scope of DNNs executed on microcontroller units (MCUs) extends to time-critical systems, it becomes important to ensure timing guarantees for increasing demand of DNN inferences. To this end, this paper proposes RT-MDM, the first Real-Time scheduling framework for Multiple DNN tasks executed on an MCU using external memory. Identifying *execution-order dependencies* among segmented DNN models and *memory requirements* for parallel execution subject to the dependencies, we propose (i) a *segment-group-based memory management policy* that achieves isolated memory usage within a segment group and sharded memory usage across different segment groups, and (ii) an *intra-task scheduler* specialized for the proposed policy. Implementing RT-MDM on an actual system and optimizing its parameters for DNN segmentation and segment-group mapping, we demonstrate the effectiveness of RT-MDM in accommodating more DNN tasks while providing their timing guarantees.

1 INTRODUCTION

Recently, the use of low-power, low-cost microcontrollers (MCUs) in IoT devices has been rapidly increasing for time-critical systems that necessitate multiple real-time DNN inference tasks, such as autonomous mini-vehicles [4]. Each MCU is usually equipped with extremely limited internal memory, which causes memory shortage issue. Since the issue cannot be completely addressed by applying existing techniques for DNN model compression and memory optimization [5, 12], a typical approach is to apply the DNN model segmentation technique [6] (for reducing the size of the DNN model loaded in the internal memory) and to employ external memory (for storing all necessary DNN models) [10, 14]. For the latter, a DMA (Direct Memory Access) device that is typically employed in the MCU enables parallel execution between *computing operations* on the CPU and *I/O operations* for data transfer between the internal and external memory. However, due to *execution-order dependencies* among segments (each performed on the DMA and then the CPU) of each DNN task and *memory requirements* for parallel execution on the CPU and DMA subject to the dependencies, it is challenging (G1) to efficiently utilize both CPU and DMA for accommodating more (segmented) DNN tasks (G2) while achieving their timing guarantees.

In this paper, we propose RT-MDM in Fig. 2, the first Real-Time scheduling framework for Multiple DNN tasks (subject to segmentation) executed on an MCU using external memory, aiming at achieving G1 and G2. Identifying the execution-order dependencies and memory requirements, we establish two system design principles for RT-MDM: (DP1) enforcing non-preemptiveness for intra-task scheduling and (DP2) proposing a new segment-group-based memory management policy. Under DP2, each segment belonging to one of the pre-defined segment groups, whenever executed, exclusively occupies the memory space designated to its segment group.

One may argue that G1 is achieved by employing a naive on-demand memory management policy, e.g., managing the memory in a best-effort manner by executing one of the ready-to-be-executed segments whose memory usage is not larger than the available memory space. However, the policy incurs an additional delay due to the priority-inversion of segments whose memory usage is different, and more importantly, it incurs a number of priority-inversion situations (e.g., combinations of a set of segments currently occupying the memory). This makes it difficult/pessimistic to identify/upper-bound the worst-case waiting time of a segment for memory usage; then, a task-level timing guarantee becomes extremely difficult/pessimistic, as we need to consider all the waiting and execution times of sequential segments belonging to a task, compromising G2.

Different from a naive policy, the proposed memory management policy DP2 provides an interface to achieve G1 and G2 *together*. This is due to its utilization of its shared memory across different segment groups (supporting G1 by identifying potential segments for parallel execution on the DMA and CPU) and isolated memory usage within each segment group (supporting G2 by limiting segment candidates that may cause priority-inversion). Additionally, DP1 makes the interface stronger in achieving G1 and G2; DP1 allows DP2 not only to tailor segment groups specifically for each task (in terms of number and size) as memory sharing is confined to segments within the same task (promoting G1), but also to further narrow down segment candidates that may incur priority-inversion into those within the same task and group (promoting G2).

From the perspective of developing the scheduler, since DP1 enables the development of the intra-task scheduler (that schedules segments of the same task) without considering the interference from other tasks in terms of both timing and memory usage, it is possible to design the intra-task scheduler and inter-task scheduler

[*]Sukmin Kang and Seongtae Lee are co-first authors.
[†]Jinkyu Lee is the corresponding author.

DAC '24, June 23–27, 2024, Moscone West, San Francisco
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655681

independently. To develop the intra-task scheduler that can benefit from the interface provided by DP2 and enhanced by DP1, we derive a new execution-order dependency specialized for the proposed memory management policy DP2 that employs DP1, and enforce an execution-order dependency that eases the computing of each task's WCET (the Worst-Case Execution Time) for the given WCET of its segments. We then apply an existing inter-task scheduler and reuse its offline timing analysis by providing the calculated task WCET as an input to the analysis.

Finally, we find a feasible parameter combination that passes the offline timing analysis subject to memory constraints, among all combinations of given segmentation and segment-group mapping for every task. Considering the trade-off between segmentation overhead and parallel execution opportunity, we formulate a global optimization problem and decompose it into smaller per-task optimization problems, enabling optimal combination identification with lower time-complexity.

We implement RT-MDM on top of TensorFlow Lite and run it on the Arduino Nano 33 BLE Sense board. Our case study with voice and gesture recognition models shows that RT-MDM, along with its optimization, successfully provides timing guarantees with/without memory constraints. To demonstrate the effectiveness of RT-MDM for various task sets with their different memory requirements, we solve the formulated optimization problem for random task sets, leading to conclusions consistent with the case study.

Related work. Due to the unique challenges originating from MCUs with limited memory space, existing studies for timing guarantees of multiple DNN tasks without memory constraints, e.g., [8, 15] cannot be directly applied to the MCU environment. On the other hand, there exist a few studies that addressed the memory constraint issue for a single DNN task or multiple DNN tasks under a non-MCU environment [6, 7]. Demand layering [7] minimizes memory usage by loading and executing DNN layers in a layer-by-layer manner, but it focuses on a single DNN task. SPET [6], which is the most relevant study to RT-MDM, focuses on timing guarantees for multiple DNN tasks on edge TPUs with limited SRAM. SPET improves schedulability performance with deadline-aware SRAM allocation and DNN partitioning but falls short in maximizing intra-task parallelism, especially crucial in tiny memory environments where even a single DNN model might not fit entirely. Note that there exist a few studies that address the delay issue for multi-DNN tasks executed on an MCU, e.g., [3]; however, they do not address timing guarantees.

Contributions. To the best of our knowledge, this paper is the first work to consider both memory constraints and timing guarantees for multiple DNN tasks executed on an MCU with external memory, making the following key contributions. Based on the investigation of execution-order dependencies and memory requirements, we establish system design principles for RT-MDM (Section 3). We develop a segment-group-based memory management policy, specialized for the target system (Section 3). We develop an intra-task scheduler for the proposed memory management policy and incorporate it into the overall scheduler and its timing analysis technique (Section 4). We formulate an optimization problem for RT-MDM and solve its decomposed problem with lower time-complexity (Section 5). We implement RT-MDM in an actual system and demonstrate its effectiveness (Section 6).

2 TARGET ARCHITECTURE AND MODEL

Computing architecture. We target an MCU that consists of a CPU and an internal memory. Since the internal memory is extremely limited, the MCU attaches external memory and deploys a DMA (Direct Memory Access) device for parallel execution between *computing operations* on the CPU and *I/O operations*. The external memory is large enough to load all necessary DNN models.

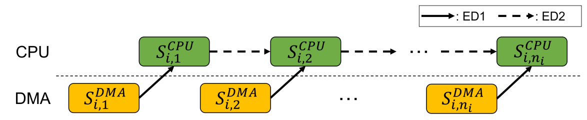


Figure 1: Execution order dependency of a DNN task τ_i

Task model. We target a set of n real-time DNN tasks executed on the MCU, denoted by $\tau = \{\tau_i\}_{i=1}^{n}$. We consider segmenting a DNN task into smaller DNN segments, making it feasible for (segmented) DNN model(s) to be loaded in the extremely limited internal memory.[1] Then, a real-time DNN task τ_i is specified by the inter-arrival time T_i, the relative deadline D_i ($\le T_i$), and a series of segments $\overrightarrow{S_i}$. Each τ_i invokes a series of instances; the release times of two consecutive instances invoked by the same task are separated by at least T_i time units, and an instance released at t should finish its execution until $t+D_i$, following the periodic/sporadic model [11]. A series of segments $\overrightarrow{S_i}$ for τ_i are represented by $\{S_{i,x}\}_{x=1}^{NS_i}$, where $S_{i,x}$ denotes the x-th segment of τ_i and NS_i denotes the number of segments for τ_i. Let $M_{i,x}$ denote the memory size requirement of the (segmented) DNN model $S_{i,x}$. Also, each $S_{i,x}$ consists of $S_{i,x}^{\mathsf{DMA}}$ and $S_{i,x}^{\mathsf{CPU}}$, which are the x-th DMA sub-segment and its paired x-th CPU sub-segment of τ_i, respectively, as shown in Fig. 1; every DMA sub-segment is non-preemptive, which also holds for every CPU sub-segment. Let $C_{i,x}^{\mathsf{DMA}}$ and $C_{i,x}^{\mathsf{CPU}}$ denote WCET of $S_{i,x}^{\mathsf{DMA}}$ and $S_{i,x}^{\mathsf{CPU}}$, respectively. Let C_i denote the sum of all segments of τ_i, i.e., $C_i = \sum_{x=1}^{NS_i} C_{i,x}^{\mathsf{DMA}} + C_{i,x}^{\mathsf{CPU}}$.

3 SYSTEM DESIGN OF RT-MDM

Specifying execution order dependencies. As shown in Fig. 1, there are inherent execution order dependencies within a DNN task: between a DMA sub-segment and its paired CPU sub-segment, and between two consecutive CPU sub-segments. The former arises because a DNN inference (which is performed by its CPU sub-segment) cannot be done without loading its DNN model in the internal memory (which is performed by its DMA sub-segment), while the latter arises because the output of a CPU sub-segment becomes the input of the next CPU sub-segment. We record the EDs (Execution-order Dependencies) as follows.

- ED1. The x-th CPU sub-segment of τ_i (i.e., $S_{i,x}^{\mathsf{CPU}}$) starts execution, only if the x-th DMA sub-segment of τ_i (i.e., $S_{i,x}^{\mathsf{DMA}}$) completes.
- ED2. The x-th CPU sub-segment of τ_i (i.e., $S_{i,x}^{\mathsf{CPU}}$) starts execution, only if the (x-1)-th CPU sub-segment of τ_i (i.e., $S_{i,x-1}^{\mathsf{CPU}}$) completes (or it is the first segment, i.e., $x = 1$).

Identifying memory requirements. We identify MRs (Memory Requirements) for parallel execution between CPU and DMA sub-segments, subject to ED1 and ED2, as follows.

- MR1. As it is not mandatory to execute $S_{i,x}^{\mathsf{DMA}}$ and $S_{i,x}^{\mathsf{CPU}}$ in a back-to-back manner, the DNN model loaded in the internal memory (by $S_{i,x}^{\mathsf{DMA}}$) should be kept not only during its inference (by $S_{i,x}^{\mathsf{CPU}}$) but also between the completion of $S_{i,x}^{\mathsf{DMA}}$ and the beginning of $S_{i,x}^{\mathsf{CPU}}$. Otherwise, we need to re-load the DNN model by re-executing $S_{i,x}^{\mathsf{DMA}}$, incurring an additional delay.

[1]Possible segmentation points vary with each DNN model, e.g., up to four segments in the gesture recognition model in Section 6. Segmentation imposes an additional delay to each segment due to the output copy operation and I/O interface preparation.

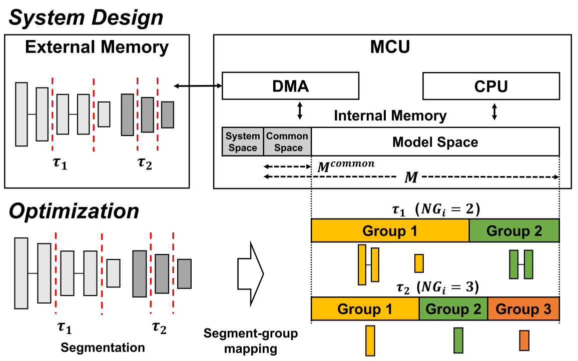


Figure 2: System design and optimization for RT-MDM

MR2. Since the output from $S_{i,x}^{\mathsf{CPU}}$ becomes an input to $S_{i,x+1}^{\mathsf{CPU}}$, we need a space in the internal memory to store the output during the interval from the completion of $S_{i,x}^{\mathsf{CPU}}$ to the beginning of $S_{i,x+1}^{\mathsf{CPU}}$. Otherwise, we need to write the output to and read it back from the external memory, incurring additional delay and management burden.

MR3. To execute a CPU sub-segment $S_{i,x}^{\mathsf{CPU}}$ and a DMA sub-segment $S_{k,y}^{\mathsf{DMA}}$ in parallel, the internal memory should accommodate the DNN models for both $S_{i,x}^{\mathsf{CPU}}$ and $S_{k,y}^{\mathsf{DMA}}$ at the same time.

Establishing design principles for RT-MDM. We establish the system Design Principles (DPs) for RT-MDM that provide an interface to achieve G1 and G2 under ED1–2 and MR1–3.

DP1. We enforce **task-level non-preemptive scheduling**; once the first segment of a task τ_i starts its execution, no segments from other tasks can be executed on both CPU and DMA until all segments of τ_i finish their executions.

DP2. We introduce a notion of segment groups for memory partition and propose **segment-group-based memory management, tailored to each task** (thanks to DP1).

Developing memory management policy. We develop our segment-group-based memory management policy, specialized for each task τ_i as follows. Segment groups are managed for each task τ_i, and each segment of τ_i belongs to one of the pre-defined segment groups for τ_i. Each segment of τ_i, whenever executed, exclusively occupies the memory space designated to its segment group.

Then, the proposed memory management policy (i.e., DP2), along with DP1, not only simplifies/addresses MR1–MR3 but also provides the scheduler (to be developed) with an interface that maximizes intra-task parallelism as follows. First, the proposed memory management policy makes the scheduler to allow the existence of multiple segments in different groups whose DMA execution is completed but CPU execution is not started, supporting parallel execution under MR1. Second, DP1 and DP2 make it possible to store only one segment output of τ_i at any time (when τ_i is executed), reducing the burden of storing segment outputs from multiple tasks. Therefore, we address MR2 by reserving a part of memory space as a common space whose size is the largest segment output and the executable space.[2] Third, we partition the memory space not reserved for the common space and system space (utilized by the system's BSP and the DNN framework) into NG_i segment groups for τ_i, where NG_i is the number of segment groups for τ_i; as shown in Fig. 2, the partition is tailored to each task τ_i. The N-th segment group of τ_i requires the memory space as much as the maximum memory usage of all segments belonging to the N-th segment group of τ_i, calculated by $\max_{S_{i,x}|G_{i,x}=N} M_{i,x}$, where $G_{i,x}$ is the segment-group index of $S_{i,x}$ ($1 \leq G_{i,x} \leq NG_i$). Then, DP1 and DP2 address MR3 by checking the following constraint for every $\tau_i \in \tau$: the sum of the maximum memory usage of all segment groups of τ_i is no larger than the available memory size the except system space (denoted by M) subtracted by the common space (denoted by M^{common}), recorded as follows.

$$\text{Constraint 1.} \quad \sum_{N=1}^{NG_i} \max_{S_{i,x}|G_{i,x}=N} M_{i,x} \leq M - M^{\text{common}} \tag{1}$$

Guiding scheduler development. DP1 and DP2 also guide the development of the scheduler for RT-MDM with several advantages. First, DP1 allows the currently executing segment of a task to operate without considering offline parameters and run-time information of segments from other tasks. This enables designing an intra-task scheduler optimized for each task independently of the inter-task scheduler. Second, DP2 reduces the burden of considering the memory usage of all segments that occupy memory space; instead, the scheduler only needs to check whether the memory space partitioned to a group is used or not, which not only reduces the run-time scheduling overhead but also simplifies possible execution scenarios, easing the development of offline timing analysis. Third, DP2 makes it possible to reuse existing schedulability analysis for the most fundamental periodic/sporadic sequential task model, as long as WCET of each task is calculated under the intra-task scheduler. According to the guidance, Section 4 will present details of the scheduler for RT-MDM that complies with ED1–2 and MR1–3, along with timing guarantees.

4 SCHEDULER DEVELOPMENT FOR RT-MDM

4.1 Intra-Task Scheduler

Focusing on a task τ_i, this section introduces an intra-task scheduling algorithm designed for a single segment group for each task τ_i (i.e., $NG_i = 1$). Then, the scheduling algorithm is generalized to multiple segment groups (i.e., $NG_i > 1$) to maximize parallel execution on the CPU and DMA.

If all segments of a task belong to the same group, the internal memory can accommodate only one (segmented) DNN model, making it impossible for any two different segments to be executed in parallel on the CPU and DMA, which enforces the following execution-order dependency:

ED3-IPD. The x-th DMA sub-segment of τ_i (i.e., $S_{i,x}^{\mathsf{DMA}}$) starts its execution, only if the (x-1)-th CPU sub-segment of τ_i (i.e., $S_{i,x-1}^{\mathsf{CPU}}$) completes (or it is the first segment, i.e., $x = 1$).

Thanks to DP1, every DMA and CPU sub-segment starts its execution as quickly as possible without considering interference from segments of other tasks, as long as ED1 and ED2 (the inherent execution-order dependencies in Section 3) and ED3-IPD (the one originating from a single segment group) are satisfied. We call this intra-task scheduling algorithm IPD (Intra-Parallelism-Disabled) scheduling. As illustrated in Fig. 3(b1), IPD executes all sub-segments in a back-to-back manner.

Different from the single segment-group case, a task with multiple segment groups allows intra-task parallel execution on the CPU and DMA. That is, during the execution of a CPU sub-segment of the task (i.e., $S_{i,x}^{\mathsf{CPU}}$), the next DMA sub-segment(s) can be executed (i.e., S_{i,y_1}^{DMA} and then S_{i,y_2}^{DMA} where $y_1, y_2 > x$), as long as those segments

[2]The common space consists of executable space and memory for segment output. Executable space is memory utilized during inference run-time, and segment output refers to the intermediate results generated by segmentation. Unlike parameter data, this common space is shared across all models, indicating that its required size is defined by the largest executable, rather than the cumulative total.

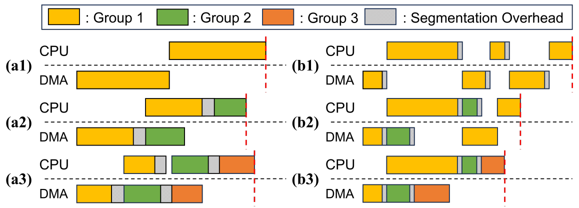


Figure 3: Intra-task schedules (a) for τ_i when NG_i=NS_i=1, 2, 3, and (b) for another task τ_j with NS_j=3 when NG_j=1, 2, 3

belong to different segment groups. This is because, there is no inherent execution order requirement between a CPU sub-segment of a task (i.e., $S^{\text{CPU}}_{i,x}$) and its next DMA sub-segments ($S^{\text{DMA}}_{i,y}$ where $y > x$). We design the following intra-task scheduling algorithm for multiple segment groups, called IPE (Intra-Parallelism-Enabled) scheduling.

First, to address the remaining memory requirement issue MR1 that was not addressed but simplified by the proposed memory management policy, we should guarantee the following execution-order dependency.

ED4-IPE. The x-th DMA sub-segment of τ_i (i.e., $S^{\text{DMA}}_{i,x}$) starts its execution, only if no prior DMA sub-segment $S^{\text{DMA}}_{i,y}$ ($y < x$) in the same group remains with its paired CPU sub-segment $S^{\text{CPU}}_{i,y}$ unfinished.

Note that the "only if" statement of ED4-IPE implicitly implies $S^{\text{DMA}}_{i,x}$ does not belong to the same group as the currently executing CPU sub-segment, which is a must for parallel execution on the CPU and DMA.

Second, to ease timing guarantees for IPE, we should avoid the inversion among consecutive DMA segments. To this end, we enforce the following execution order requirement to IPE, to be used for Lemma 2 that calculates the task WCET under IPE.

ED5-IPE. The x-th DMA sub-segment of τ_i (i.e., $S^{\text{DMA}}_{i,x}$) starts its execution, only if the (x-1)-th DMA sub-segment of τ_i (i.e., $S^{\text{DMA}}_{i,x-1}$) completes (or it is the first segment, i.e., $x = 1$).

While ED4-IPE implicitly enforces sequential DMA execution for segments in the same group, ED5-IPE applies to all segments.

Similar to IPD, every DMA and CPU sub-segment under IPE starts its execution as quickly as possible, as long as necessary execution order requirements are satisfied. Therefore, under IPE, each CPU sub-segment starts as soon as the inherent execution order requirements ED1 and ED2 are satisfied, while each DMA sub-segment starts as soon as ED4-IPE and ED5-IPE are satisfied.

Note that IPE is a generalization of IPD, as IPD is equivalent to IPE when all segments of a task belong to the same group. This is because, if all segments belong to the same segment group, ED4-IPE is not satisfied for any DMA sub-segment during any CPU sub-segment execution, yielding ED3-IPD that implies ED5-IPE.

We illustrate various intra-task schedules under IPE in Fig. 3, where NS_i and NG_i are the number of segments and segment groups of τ_i, respectively. Fig. 3(a) shows the impact of finer segmentation on the task WCET, when memory space is sufficiently large to accommodate DNN models for all segments (i.e., one group allocated to each segment). Due to the segmentation overhead shown in the gray box, finer segmentation does not always yield a smaller task WCET. On the other hand, Fig. 3(b) depicts a varying number of segment groups, for a given segmentation. By increasing the opportunity for parallel execution, more segment groups always yield a shorter task WCET, as long as Constraint 1 is satisfied.

Note that it is possible for two consecutive segments to be assigned to the same group in order to satisfy Constraint 1.

4.2 Overall Scheduler with Timing Analysis

While DP1 enforces NP (Non-Preemptiveness) for inter-task scheduling (i.e., the execution of a task's segment does not allow that of any segment of other tasks), we can apply most (if not all) existing prioritization policies for inter-task scheduling such as FP (task-level Fixed-Priority) and EDF (Earliest Deadline First). In this paper, we focus on NP-FP-IPD and NP-FP-IPE as follows: to determine which task to be executed, we enforce pre-defined task-level fixed priority (FP); once a segment of a task starts its execution, it is not interfered by any segment from other tasks (NP); and once a task is selected to be executed, its segments are scheduled by IPE (or IPD). Let $\mathsf{HP}(\tau_k)$ and $\mathsf{LP}(\tau_k)$ denote a set of tasks higher and lower than τ_k, respectively for FP.

Therefore, as long as we calculate WCET of each task under the intra-task scheduler, the execution model for RT-MDM is abstracted to the basic sporadic/periodic model on a uniprocessor [11]. Under NP-FP-IPD, WCET of each task τ_i (denoted by C'_i) is trivially the sum of WCET of all CPU and DMA sub-segments, i.e., $C'_i = C_i = \sum_{x=1}^{NS_i} C^{\text{DMA}}_{i,x} + C^{\text{CPU}}_{i,x}$. Then, we can apply an existing response time analysis for the basic sporadic/periodic model, yielding the following response time analysis for NP-FP-IPD.

Lemma 1. *[From [9]] Suppose the segmentation $\overrightarrow{S_i}$ and segment group mapping $\{G_{i,x}\}_{x=1}^{NS_i}$ are given for every $\tau_i \in \tau$, and the model space in the internal memory is sufficiently large to accommodate a DNN model for any segment. A DNN task set τ is schedulable by NP-FP-IPD, if the response time $(F_k + C'_k - 1)$ is no larger than D_k for every $\tau_k \in \tau$, where $F_k = L$ that satisfies the following inequality:*

$$1 + \max_{\tau_j \in \mathsf{LP}(\tau_k)} \left(C'_j - 1\right) + \sum_{\tau_h \in \mathsf{HP}(\tau_k)} W_h(L) \leq L, \tag{2}$$

where $W_i(L)$ is the maximum amount of execution of τ_i in an interval of length L (therefore upper-bounded by L) calculated as follows [1]. Note that $N_i(L) = \lfloor (L + R_i - C'_i)/T_i \rfloor$, and the fixed-point iteration (e.g., used in [9]) is used to find L for Eq. (2).

$$W_i(L) = N_i(L) \cdot C'_i + \min\left(C'_i, L + R_i - C'_i - N_i(L) \cdot T_i\right) \tag{3}$$

Proof. We outline the proof, with the full version in [9]. If Eq. (2) holds, one unit of τ_k's execution is finished within the interval of length L regardless of blocking from a lower-priority task (upper-bounded by $\max_{\tau_j \in \mathsf{LP}(\tau_k)} (C'_j - 1)$) and interference from higher-priority tasks (upper-bounded by $\sum_{\tau_h \in \mathsf{HP}(\tau_k)} W_h(L)$). Due to non-preemptiveness, τ_k finishes within $L + C'_k - 1$ time units. □

Different from IPD, we need to develop a method to calculate WCET of τ_i (denoted by C^*_i) under IPE.

Lemma 2. *Suppose the segmentation $\overrightarrow{S_i}$ and segment group mapping $\{G_{i,x}\}_{x=1}^{NS_i}$ are given for every $\tau_i \in \tau$, and the model space in the internal memory is sufficiently large to accommodate DNN models of any multiple segments belonging to all different segment groups. Under IPE, for every $\tau_i \in \tau$, the following holds: once τ_i starts its execution, it finishes its execution within C^*_i time units, where C^*_i is calculated by Algorithm 1.*

Proof. Algorithm 1 is equivalent to simulating IPE from $t = 0$ and taking the finishing time as C^*_i, when the actual execution time of each sub-segment is equal to its WCET.

Algorithm 1 Calculation of C_i^* for τ_i under IPE

```
1: $t^{\mathsf{DMA}} \leftarrow \mathtt{INF}$, $t^{\mathsf{CPU}} \leftarrow \mathtt{INF}$, $x \leftarrow 1$, $y \leftarrow 1$, $t \leftarrow 0$, Pending$\leftarrow \emptyset$
2: while $x \le NS_i$ or $y \le NS_i$ do
3:     if $t^{\mathsf{DMA}} = \mathtt{INF}$ and $G_{i,x} \notin$ Pending then $t^{\mathsf{DMA}} \leftarrow t + C_{i,x}^{\mathsf{DMA}}$,
       Pending.add($G_{i,x}$) // This line is skipped when $x = NS_i + 1$
4:     if $t^{\mathsf{CPU}} = \mathtt{INF}$ and $y < x$ then $t^{\mathsf{CPU}} \leftarrow t + C_{i,y}^{\mathsf{CPU}}$
5:     $t = \min(t^{\mathsf{DMA}}, t^{\mathsf{CPU}})$
6:     if $t = t^{\mathsf{DMA}}$ then $x \leftarrow x + 1$, $t^{\mathsf{DMA}} \leftarrow \mathtt{INF}$
7:     if $t = t^{\mathsf{CPU}}$ then Pending.remove($G_{i,y}$), $y \leftarrow y + 1$, $t^{\mathsf{CPU}} \leftarrow \mathtt{INF}$
8: end while
9: Return $t$ as $C_i^*$
```

In Line 1, t^{DMA} and t^{CPU} denote the finishing time of the currently executing DMA and CPU sub-segments, respectively, with INF indicating no currently executing corresponding sub-segment. x and y denote the index of the DMA and CPU sub-segments to be executed, respectively; t is the current time, and Pending is a set of segment group indices with finished DMA but unfinished paired CPU sub-segments (needed for checking ED4-IPE). The while loop in Lines 2–8 simulates IPE from $t = 0$, with the finishing time returned in Line 9. Lines 3–4 execute the next DMA (likewise CPU) sub-segment if the necessary conditions are satisfied; Line 5 updates the current time after executing DMA and/or CPU sub-segment(s); and Lines 6–7 update necessary information when the currently executing DMA and/or CPU sub-segment(s) is finished.

Then, Algorithm 1 yields each task's WCET under IPE, as IPE enforces sequential execution of CPU sub-segments (by ED2) and DMA sub-segments (by ED5-IPE); by preventing any inversion among CPU sub-segments and that among DMA sub-segments, Algorithm 1 is sustainable with respect to the actual execution time of every segment, which proves the lemma. This implies that, under IPE without ED5-IPE, Algorithm 1 does not yield the task WCET, making it very difficult (or impossible) to calculate the task WCET. □

It is straightforward that C_i^* in Lemma 2 is no larger than $C_i' = C_i$. Therefore, under the same prioritization policy for inter-task scheduling, IPE yields no larger response time of every $\tau_i \in \tau$ than IPD. Using Lemma 2, we derive a response time analysis for NP-FP-IPE, which is tighter than that for NP-FP-IPD in Lemma 1.

Theorem 1. *Suppose the segmentation $\overrightarrow{S_i}$ and segment group mapping $\{G_{i,x}\}_{x=1}^{NS_i}$ are given for every $\tau_i \in \tau$, and the model space in the internal memory is sufficiently large to accommodate DNN models of any multiple segments belonging to all different segment groups. A DNN task set τ is schedulable by NP-FP-IPE, if the response time ($F_k^* + C_k^* - 1$) is no larger than D_k for every $\tau_k \in \tau$, where $F_k^* = L$ that satisfies Eq. (2) by replacing all $\{C_i'\}_{\tau_i \in \tau}$ terms to $\{C_i^*\}_{\tau_i \in \tau}$.*

Proof. The theorem immediately holds by Lemmas 1 and 2. □

5 OPTIMIZATION FOR RT-MDM

So far, we have explained how RT-MDM operates at run-time once the segmentation (i.e., $\overrightarrow{S_i}$) and the segment-group mapping (i.e., $\{G_{i,x}\}_{x=1}^{NS_i}$)[3] are given for every $\tau_i \in \tau$. However, the choice of the segmentation and segment-group mapping (determined offline) affects timing guarantees in Theorem 1 due to a trade-off between segmentation overheads and parallel execution opportunities. Finer segmentation yields larger overhead (in time), due to the time needed for saving/restoring intermediate output data from the CPU and preparing DMA protocols. However, it enables the use of more segment groups, which enhances opportunities for parallel execution, thereby reducing task WCET. Considering this trade-off, we find an optimal segmentation and segment-group mapping (shown in Fig. 2) by solving the following target problem.

Time (ms) Size (KB)	τ_1 NS_1=1	τ_1 NS_1=2	τ_2 NS_2=1	τ_2 NS_2=2	τ_2 NS_2=3	τ_2 NS_2=4
Max $\{C_{i,x}^{\mathsf{DMA}}\}$	98	11,89	115	16,110	14,103,16	14,31,81,14
Max $\{C_{i,x}^{\mathsf{CPU}}\}$	187	203,11	186	144,51	145,46,6	146,24,22,6
$\{M_{i,x}\}$	27	3,25	31	3,30	9,22,3	3,7,22,3

Table 1: Measurement of $\{C_{i,x}\}$ and $\{M_{i,x}\}$ for τ_1 and τ_2

Problem 1 (Global problem). *For every $\tau_i \in \tau$, the options for $\overrightarrow{S_i}$ are given[4], and the options for each $G_{i,x}$ for $1 \le x \le NS_i$ are $1 \le G_{i,x} \le NS_i$. Among all combinations of the options for every $\tau_i \in \tau$, **Find** a combination that satisfies Theorem 1 subject to Constraint 1 in Eq.* (1).

Since the search space for Problem 1 is too broad (i.e., tasks × segmentation options × segment-group mapping options), we decompose the problem into the following smaller per-task optimization problems.

Problem 2 (Decomposed problem for τ_i). *For given options for $\overrightarrow{S_i}$ and options for $\{1 \le G_{i,x} \le NS_i\}_{x=1}^{NS_i}$, **Minimize** C_i^* subject to Constraint 1 in Eq.* (1).

We can solve Problem 2 using existing optimization solvers, e.g., [13]. Finally, the following theorem proves that solving the decomposed per-task problems for all tasks one-by-one conserves the optimality of Problem 1,

Theorem 2. *If Theorem 1 does not hold for $\{C_i^*\}$ solved by Problem 2 for every $\tau_i \in \tau$, there is no solution for Problem 1.*

Proof. DP1 disallows any dependency among different tasks, and Theorem 1 is sustainable with respect to C_i^* (i.e., smaller C_i^* always reducing or at least keeping the response time of every task), which proves the theorem. □

6 EVALUATION

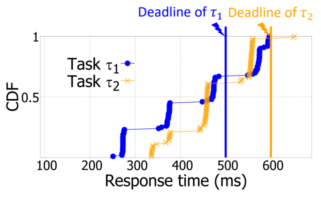
Implementation and Case Study. We implement RT-MDM on the Arduino Nano 33 BLE Sense board consisting of an ARM Cortex M4F core that runs at 64 MHz with a Nordic nRF52480 chipset, a compact 256KB SRAM, and a DMA device; an SD card is attached to the board as external memory. We develop RT-MDM on top of TensorFlow Lite without any additional modifications.

We trained and implemented two commonly used types of DNN models on MCUs: voice recognition (for some commands) and gesture recognition (using accelerometer data), which were created by combining conv2d layers and fully connected layers. The voice and gesture models without any segmentation are 27KB and 31KB in size, respectively. The voice recognition model (denoted as τ_1) could be divided into up to two segments, while the gesture recognition model (denoted as τ_2) could be segmented into at most four segments, each with its own segmentation point. Table 1 presents the maximum of the 100 measured execution times for all segments of τ_1 and τ_2, for all possible combinations of segmentation. Note that, for a given NS_i, we show just one segmentation option among all possible combinations within a task.

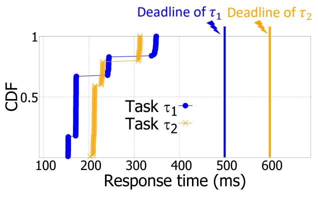
We set the remaining parameters of τ_1 and τ_2 as T_1=D_1=500ms and T_2=D_2=600ms, and compare the actual response time (by measurement) and guaranteed response time (by Theorem 1), under the followings: IPE-1-1 (IPE with 1 segment and 1 segment group,

[3]Note that NG_i is implicitly determined, once $\{G_{i,x}\}_{x=1}^{NS_i}$ is determined.

[4]Each DNN model has its own possible segmentation points. See examples in Section 6.



(a) Under IPE-M-1



(b) Under IPE-Opt

Figure 4: The actual response time for case study

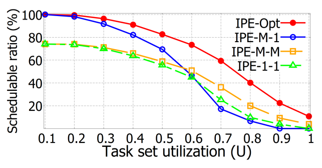
which is equivalent to IPD), IPE-M-1 (IPE with the maximum segments and 1 segment group), IPE-M-M (IPE with the maximum segments and the maximum segment groups), and IPE-Opt (IPE with the optimal segment and segment group that yields the solution of Problem 2). Note that we employ NP-DM (Deadline Monotonic; the smaller D_i, the higher priority) for the inter-task scheduler.

If the model space in the internal memory is sufficiently large, the actual response times of both tasks under IPE-Opt and IPE-M-M do not exceed the corresponding relative deadline D_i, e.g., IPE-Opt in Fig. 4(b). However, those under IPE-M-1 and IPE-1-1 compromise timing constraints (e.g., the largest response times under IPE-M-1 in Fig. 4(a) are 596ms for τ_1 and 651ms for τ_2), due to incapability of parallel execution. The schedulability results are consistence with the guaranteed response times from Theorem 1; for example, the guaranteed response time of τ_2 under IPE-Opt is 436ms ($\leq D_2$=600ms), while that under IPE-M-1 is not bounded.

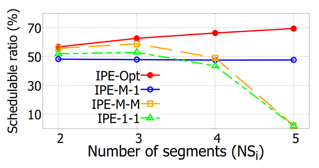
However, if the model space in the internal memory is limited to 30KB, IPE-M-M no longer yields timing guarantee, due to compromising Constraints 1, i.e., the sum of $\{M_{2,x}\}$ when $NS_2 = NG_2 = 4$ in Table 1 is 3+7+22+3=35KB, larger than 30KB. This holds for all approaches with $NG_2 = NS_2$, because the sum of $\{M_{i,x}\}$ for every column for τ_2 in Table 1 is larger than 30KB. On the other hand, IPE-Opt chooses $NS_2 = 4$ and two segment groups ($NG_2 = 2$) of $\{S_{2,1}, S_{2,3}\}$ and $\{S_{2,2}, S_{2,4}\}$, which requires only max(3,22)+max(7, 3)=29KB memory space and yields the guaranteed response time of 494ms ($\leq D_2$=600ms). This demonstrates that RT-MDM, along with its optimization, not only considers memory constraints but also ensures timing guarantees.

More detailed evaluation for timing guarantee. To evaluate the capability of RT-MDM in achieving timing guarantees subject to the memory constraint for *various task sets with their different memory requirements*, we count task sets deemed schedulable by Theorem 1 with Constraint 1, among task sets randomly generated as follows. For given task set utilization (U=$\sum_{\tau_i\in\tau}\frac{C_i}{T_i}$ = $0.1, 0.2, 0.3, \cdots, 1.0$), we choose the number of DNN tasks (n) in [2, 5] (four options), and then set the period T_i (the same as the relative deadline D_i) of each task uniformly from [5000, 50000). The utilization of each task (u_i=$\frac{C_i}{T_i}$) is generated with the UUnifast algorithm [2], and the corresponding C_i is set to $u_i \times T_i$. For $|\overrightarrow{S_i}|$, the number of segments of a task τ_i (NS_i) is selected in [2,5] (four options); we then distribute C_i to all sub-segments of τ_i, i.e., $\{C^{\text{DMA}}_{i,x}, C^{\text{CPU}}_{i,x}\}$, using UUnifast. We generate segmentation overhead from $[0.1 \cdot C_{i,x}, 0.2 \cdot C_{i,x}]$, and add it to each sub-segment of $S_{i,x}$. For given model space in the internal memory (i.e., $M - M^{\text{common}}$), $M_{i,x}$ is uniformly selected in $[0.1 \cdot M, 0.3 \cdot M]$. For each target task set utilization U, we generate 1,000 task sets.

Fig. 5(a) shows the schedulable task set ratio for 1000×4×4 task sets per task set utilization (U) under different approaches. Overall, IPE-Opt exhibits 32.0%, 45.7% and 60.5% higher schedulable ratio than IPE-1-1, IPE-M-M and IPE-M-1, respectively. Since IPE-1-1 and IPE-M-M enforce the memory constraint where the sum of all segments' memory usage is less than the model space in the



(a) Varying U



(b) Varying NS_i for given $n = 5$

Figure 5: The ratio of schedulable task sets

internal memory, their schedulable ratio is not close to 100% even with very low task set utilization. The difference between IPE-Opt and IPE-M-1 comes from our optimization strategy, which proves that it is more effective than the strategy that simply maximizes intra-parallelism. Fig. 5(b) shows the schedulable task set ratio as NS_i changes when n is fixed to 5. As NS_i increases, the schedulable ratio under IPE-M-M and IPE-1-1 decreases due to the memory constraint, but that of under IPE-M-1 remains stable; however, no approaches yield a higher schedulable ratio than IPE-Opt. In summary, IPE-Opt *always* yields a higher ratio of schedulable task sets, demonstrating the effectiveness of optimized RT-MDM.

7 CONCLUSION

This paper proposed RT-MDM, the first real-time scheduling framework for multiple DNN tasks executed on an MCU. With a novel design of the segment-group-based memory management policy and the intra-task scheduler, RT-MDM was proven to achieve efficient utilization of both CPU and DMA for accommodating more DNN tasks while providing their timing guarantees.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1A4A3018824, RS-2023-00248143, 2018R1A5A1060031 (ERC), RS-2023-00213309).

REFERENCES

[1] M. Bertogna and M. Cirinei. 2007. Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms. In *IEEE RTSS*.
[2] E. Bini and G. Buttazzo. 2005. Measuring the performance of schedulability tests. *Real-Time Systems* (2005).
[3] B. Cox, J. Galjaard, A. Ghiassi, R. Birke, and L. Y. Chen. 2021. Masa: Responsive Multi-DNN Inference on the Edge. In *IEEE PerCom*.
[4] M. D. Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos. 2021. Robustifying the Deployment of tinyML Models for Autonomous Mini-Vehicles. *Sensors* 21, 4 (2021).
[5] Y. Gong, L. Liu, M. Yang, and L. Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).
[6] C. Han, H. Chwa, K. Lee, and S. Oh. 2023. SPET: Transparent SRAM Allocation and Model Partitioning for Real-time DNN Tasks on Edge TPU. In *ACM DAC*.
[7] M. Ji, S. Yi, C. Koo, S. Ahn, D. Seo, N. Dutt, and J. Kim. 2022. Demand Layering for Real-Time DNN Inference with Minimized Memory Usage. In *IEEE RTSS*.
[8] W. Kang, K. Lee, J. Lee, I. Shin, and H. Chwa. 2021. LaLaRAND: Flexible Layer-by-Layer CPU/GPU Scheduling for Real-Time DNN Tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*.
[9] J. Lee and K. Shin. 2014. Improvement of Real-Time Multi-Core Schedulability with Forced Non-Preemption. *IEEE Transactions on Parallel and Distributed Systems* 25, 5 (2014).
[10] H. Miao and F. X. Lin. 2022. Towards Out-of-core Neural Networks on Microcontrollers. In *IEEE/ACM Symposium on Edge Computing*.
[11] A. Mok. 1983. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Ph. D. Dissertation. MIT.
[12] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).
[13] Laurent Perron and Vincent Furnon. 2023. *OR-Tools*. Google. https://developers.google.com/optimization/
[14] S. Sadiq, J. Hare, P. Maji, S. Craske, and G. V. Merrett. 2022. Tinyops: Imagenet scale deep learning on microcontrollers. In *IEE/CVF CVPR*.
[15] Y. Xiang and H. Kim. 2019. Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference. In *IEEE RTSS*.