Title

DREAM: A Dynamic Scheduler for Dynamic Real-time Multi-model ML Workloads

Authors

Kim, Seah

Kwon, Hyoukjun

Song, Jinook

et al.

Publication Date

2023-03-25

DOI

10.1145/3623278.3624753

DREAM: A Dynamic Scheduler for Dynamic Real-time Multi-model ML Workloads

Seah Kim[*]
UC Berkeley
Berkeley, CA, USA
seah@berkeley.edu

Hyoukjun Kwon[†]
UC Irvine, Meta
Irvine, CA, USA
hyoukjun.kwon@uci.edu

Jinook Song
Meta
Sunnyvale, CA, USA
jinooksong@meta.com

Jihyuck Jo
Meta
Sunnyvale, CA, USA
jjo@meta.com

Yu-Hsin Chen
Meta
Sunnyvale, CA, USA
yhchen@meta.com

Liangzhen Lai
Meta
Sunnyvale, CA, USA
liangzhen@meta.com

Vikas Chandra
Meta
Sunnyvale, CA, USA
vchandra@meta.com

ABSTRACT

Emerging real-time multi-model ML (RTMM) workloads such as AR/VR and drone control involve dynamic behaviors in various granularity; task, model, and layers within a model. Such dynamic behaviors introduce new challenges to the system software in an ML system since the overall system load is not completely predictable, unlike traditional ML workloads. In addition, RTMM workloads require real-time processing, involve highly heterogeneous models, and target resource-constrained devices. Under such circumstances, developing an effective scheduler gains more importance to better utilize underlying hardware considering the unique characteristics of RTMM workloads. Therefore, we propose a new scheduler, DREAM, which effectively handles various dynamicity in RTMM workloads targeting multi-accelerator systems. DREAM quantifies the unique requirements for RTMM workloads and utilizes the quantified scores to drive scheduling decisions, considering the current system load and other inference jobs on different models and input frames. DREAM utilizes tunable parameters that provide fast and effective adaptivity to dynamic workload changes. In our evaluation of five scenarios of RTMM workload, DREAM reduces the overall UXCost, which is an equivalent metric of the energy-delay product (EDP) for RTMM defined in the paper, by 32.2% and 50.0% in the geometric mean (up to 80.8% and 97.6%) compared to state-of-the-art baselines, which shows the efficacy of our scheduling methodology.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; **Heterogeneous (hybrid) systems**.

KEYWORDS

Scheduler, AR/VR, Multi-model ML, Hardware-Software Co-Design

ACM Reference Format:
Seah Kim, Hyoukjun Kwon, Jinook Song, Jihyuck Jo, Yu-Hsin Chen, Liangzhen Lai, and Vikas Chandra. 2023. DREAM: A Dynamic Scheduler for Dynamic Real-time Multi-model ML Workloads. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3623278.3624753

1 INTRODUCTION

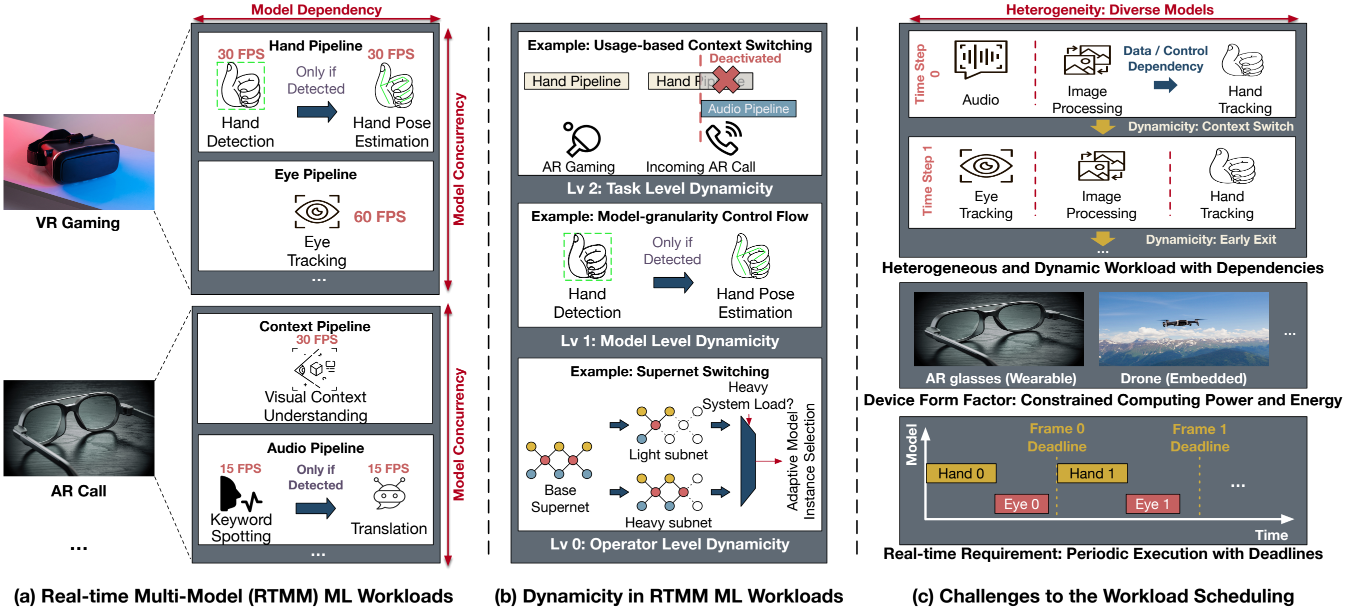
As ML-based applications become more diverse, ML inference workloads in emerging applications such as augmented and virtual reality (AR/VR) deploy many ML models with complex dependency and concurrency [17], as shown in Figure 1 (a). Such applications often require real-time processing, which lead to real-time multi-model (RTMM) ML inference workloads (e.g., drone navigation [32]). RTMM workloads impose unique requirements that distinguish them from previous ML inference workloads often based on one or several independent models without model level dependency and concurrency [29]. As summarized in Figure 1 (c), such challenges include (1) highly heterogeneous ML models (e.g., model size, operators, and tensor size) from diverse tasks and multi-modal sensor inputs, (2) rich dynamicity in various levels as illustrated in Figure 1 (b), (3) complex model level data and control dependencies, (4) constrained computing power and energy in target devices (e.g., AR glasses), and (5) real-time requirements due to continuous and periodic processing of tasks with deadlines.

Among these challenges, dynamic workloads can easily lead to unpredictable system loads, imposing a new challenge for ML systems that previously relied on highly deterministic latency information for scheduling decisions [8, 16]. In addition, the dynamicity

[*] Work done during Summer Internship at Meta
[†] Corresponding author

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0394-2/23/03…$15.00
https://doi.org/10.1145/3623278.3624753



(a) Real-time Multi-Model (RTMM) ML Workloads (b) Dynamicity in RTMM ML Workloads (c) Challenges to the Workload Scheduling

Figure 1: A summary of the motivation for this work. (a) Example real-time multi-model ML workloads that have multiple concurrent pipelines and cascaded models within some pipelines, which adds control and data dependencies to the scheduling consideration. (b) Three levels of dynamicity found in real-time multi-model workloads and examples of each. (c) Challenges to the scheduler from workloads and dynamicity.

exists in various levels of granularity; across task (i.e., which models to include in the workload), model (i.e., which version of a model to run [4, 41]), and operator (i.e., which layers to run within a model) levels. Combined with the real-time requirements on constrained hardware resources, diverse dynamicity becomes a non-trivial challenge for real-time ML systems, in particular for schedulers. Although some static scheduling methods have been proposed [16] for multi-model workloads, they are not tailored for the RTMM workloads. Prior works have proposed dynamic scheduling-based methods, which shed light on a part of the challenges targeting multi-tenancy [6, 8, 12, 21] within a task. However, they are not tailored for the dynamicity of RTMM workloads and other unique challenges of RTMM workloads, as summarized in Table 1.

To address new challenges from RTMM, we propose DREAM (Scheduler for **D**ynamic and **Rea**l-time **M**ulti-model Multi-Task ML Workloads), which targets ML accelerator-based systems and holistically considers all the main challenges of emerging RTMM workloads: real-time requirements (FPS, deadlines), concurrent processing of multiple tasks with cascaded models, and adapting to dynamic workload changes. For the real-time processing and concurrency challenge, we propose a score metric named MapScore that considers both urgency and fairness, which facilitates optimization not only for task-specific performance but also for overall performance across all tasks. For the complex dependency challenge of cascaded models, DREAM tracks the model dependency within an input frame and across multiple frames. For the dynamicity challenge, we develop a dynamic scheduling method with tunable parameters that provides fast and effective adaptivity to workload changes. DREAM also supports a variety of ML systems based on accelerators, even ones including multiple accelerators with heterogeneous size and dataflow like heterogeneous dataflow accelerators proposed in Herald [16].

We evaluate DREAM using UXCost introduced in Section 3.5, which attempts to quantify the overall user experience using deadline violation rate and energy consumption. On average, DREAM reduces UXCost by 32.2% (up to 80.8%) and 50.0% (up to 97.6%) compared to state-of-the-art baselines Planaria [8] and Veltair [21], respectively, on industry-originated RTMM workloads [17, 32].

We summarize our contributions as follows:

- A score tailored for driving scheduling decisions, which thoroughly captures key requirements and unique characteristics of RTMM workloads.
- A dynamic scheduler with tunable parameters and an online tuning method that provides fast adaptivity for workload changes without blocking the execution of workloads.
- A preemptive frame drop method that proactively drops a frame early when a deadline violation is expected, which facilitates global optimization across frames and models.
- Exploration of Supernet switching [4] in the context of RTMM that leverages a weight-sharing Supernet to improve ML system schedulers by dynamically switching to lighter model variants under heavy system loads, which also facilitates the optimization in a global scope.
- Case studies on both homogeneous and heterogeneous hardware accelerator systems with different sizes and dataflows running industry-originated realistic RTMM ML workloads, which provide new insights on the scheduling problem for RTMM workloads.

2 BACKGROUND AND MOTIVATION

Multi-task multi-model ML workloads have emerged from complex applications utilizing ML for various sub-tasks. Among them, applications such as AR/VR and autonomous driving require real-time processing, which constructed a new class of ML workload, real-time multi-model (RTMM) ML workloads [17]. We discuss their unique characteristics and implications for the ML system design.

2.1 Characteristics of RTMM Workloads

Traditional ML workloads run inferences on a single model for a single user or a collection of different single-model workloads for many users (i.e., multi-tenancy). Unlike such workloads, RTMM workloads require to (1) run multiple models in a cascaded manner with inter-model dependencies (e.g., hand pipeline in Figure 1 (a)), (2) concurrently run multiple ML pipelines, (3) meet real-time requirements (i.e., meeting deadlines for periodic inferences), and (4) support dynamic workloads that change based on user inputs or user context changes. We discuss each challenge in detail as follows.

Cascaded models (ML pipeline). ML pipelines, which cascade multiple models to perform more sophisticated tasks (e.g., eye tracking pipeline: cascaded eye segmentation and gaze estimation models [48]), have emerged to solve complex ML problems. Such ML pipelines with cascaded models add dependency across multiple models, which is one of the key differences from traditional multi-tenancy ML workloads.

Concurrent ML pipelines. Complex applications such as AR are based on diverse tasks. For example, a VR game can require both hand- and eye-tracking pipelines concurrently to provide a highly immersive and interactive user experience [17]. That is, hand- and eye-tracking pipelines need to be executed concurrently for such an application, which introduces the concurrency challenge.

Real-time Requirement. Applications that involve active interaction with a user (e.g., VR) or environment (e.g., drone control) require real-time processing of ML workloads. The real-time requirement can be translated into three core system requirements: periodically streamed input data, target processing rate (FPS), and processing deadline for each frame.

Dynamicity. Another notable characteristic of emerging RTMM workloads is the dynamicity: the workload changes based on the user inputs and environment. For example, if a drone flying in a building moves out from the building, the navigation ML model should be updated from an indoor environment-oriented to a model optimized for outdoor environments. Such changes can occur at diverse granularity, including at task (when the task changes, the model changes accordingly), model (model cascade pipeline with dependency), and operator (Supernet model variant, branchy early-exit behavior) levels, as illustrated in Figure 1 (b). As we identify the dynamicity as one of the core challenges in RTMM workloads, we discuss the dynamicity in detail next.

2.2 Dynamicity in Workloads

We introduce three levels of dynamicity found in RTMM workloads and discuss why the static scheduling approach would fail due to such dynamicity. The sources of dynamicity in the ML workloads range from the intra-model to the task levels within an RTMM workload, as illustrated in Figure 1 (b).

Task Level Dynamicity. As the RTMM workload is a real-time workload, the user context and usage scenario can change over time. For instance, a user playing a hand-interaction-based VR game would completely change the usage scenario to an AR call when the user receives an incoming AR call. Such a usage scenario change triggers a context switch to a totally different set of ML pipelines and models. In such a case, the new pipeline has to be scheduled immediately, and the rest of the models in the pipeline need to be flushed or dropped. This is a major challenge to static algorithm-based schedulers because they need to stop the execution and reconstruct an entire schedule every time a new set of workloads is identified.

Model Level Dynamicity. In ML pipelines that include cascaded models, the execution of a model in an ML pipeline depends on the results of prior models of the ML pipeline. When the dependency is control dependency, the workload becomes non-deterministic as the results are only available after running the prior model and vary based on inputs to the ML pipeline. For such non-deterministic workloads, it is infeasible for static schedulers to generate a valid and optimized schedule in advance as the actual workload is unknown at the scheduling time.

Operator Level Dynamicity. Supernet [41] refers to an emerging class of models that have large base model structures where their subsets (i.e., sub-network) are selected for different deployment scenarios. Supernet facilitates the training of multiple models with a single training process, which provides scalability for the model development process. Recent works such as Once-for-all [4] utilize the Supernet approach to train multiple versions of a model in the model size - model performance (e.g., accuracy) trade-off space. For the ML system, such an approach allows an optimization technique to select the best version of a Supernet-based model depending on the system status (e.g., overall system load, thermal, etc.) and application requirements (e.g., face recognition for unlocking a phone requires very high accuracy). Although such an emerging optimization technique is an effective approach, it introduces another dynamicity within a model, which increases the complexity of the scheduling problem.

Unlike previous work that selects the model instance offline [38], we explore a new way to utilize Supernet-based models [4] for better scheduling decisions in the RTMM context, which we discuss in detail in Section 4.5.1. Beyond Supernet-based models, dynamicity can originate from models utilizing control flow-based techniques to select the best path based on intermediate scores. Examples include early-exit [14, 36] and layer-skipping [40, 42, 45] methods. The goal of those models is to select the optimal computation-graph traversal in the accuracy-compute overhead trade-off space. Due to their dynamic nature, static scheduling would not be able to leverage such techniques, leading to conservative scheduling targeting worst cases (i.e., the longest path) to ensure correctness.

2.3 Limitation of Static Scheduling

Figure 2 shows the deadline violation rate of static and dynamic first-come-first-served (FCFS) schedulers using the `AR_Call` workload in four accelerator styles listed in Table 2. We select the `AR_Call`

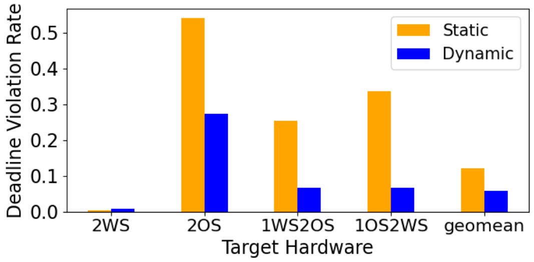


Figure 2: The deadline violation rate on `AR_Call` workload used in the evaluation (Table 3) using static and dynamic first-come-first-served (FCFS) schedulers.

scenario as it has both an audio pipeline and a dynamic model, SkipNet [42]. We assume a 50% probability for the positive keyword spotting results in the audio pipeline and a 50% probability of skipping layers for SkipNet, which provides 72% of Top-1 accuracy on ImageNet [42].

As Figure 2 shows, even if we apply the same simple FCFS algorithm, dynamic scheduling decreases the deadline violation rate by 52.9%, on average. This presents a good motivation for designing a dynamic scheduler for RTMM workloads. Although we focused on the deadline violation rates considering the real-time requirements, there are many other factors to consider to design an effective dynamic scheduler, which includes energy, hardware heterogeneity, and usage scenario-scope optimization (not local optimization for each model). To holistically consider all the aspects, we define a set of scoring metrics and utilize the metric to drive scheduling decisions. We discuss the details of our scoring metric next.

3 SCORING METRICS

In this section, we introduce our scoring metric that considers (1) **Urgency**, the time margin to a deadline modeling real-time requirements, (2) **Preference**, hardware heterogeneity and their preference to different ML operators (or layers) modeling the heterogeneity, (3) **Starvation**, the degree of starvation of each model, and (4) **Energy**, estimated energy consumption for running an operator that considers the constrained energy in target devices of RTMM workloads (e.g., AR glasses). Based on the unit score metrics, we introduce MapScore as a comprehensive metric that captures all the important requirements considered in the four unit scores.

Algorithm 1 MapScore computation for an inference task

```
▷ Inputs> $\boldsymbol{acc}$(accelerator ID), $\boldsymbol{tsk}$ (inference task ID), $\boldsymbol{AccList}$ (A list of accelerators), $\boldsymbol{N_{acc}}$ (The number of accelerators), $\boldsymbol{T_{curr}}$ (current time), $\boldsymbol{T_{cmpl}}$ (A list of the lastly scheduled layer completion time for each task), $\boldsymbol{Q_{task}}$ (Queues for remaining layers for each task), $\boldsymbol{T_{deadline}}$ (A list of deadlines for each task), $\boldsymbol{Est_{Latency}}$ and $\boldsymbol{Est_{Energy}}$ (Estimated latency and energy for each accelerator and layer pair generated offline by a cost model or a simulator.), $\boldsymbol{Stack_{task}}$ (stacks for each task tracking the completed layers and accelerators executed each layer)
▷ Output> MapScore($tsk$, $acc$): MapScore for $tsk$ on accelerator $acc$
1: Function MapScore($tsk$, $acc$, AccList, $\mathrm{T_{curr}}$, $\mathrm{T_{cmpl}}$, $\mathrm{T_{deadline}}$, $\mathrm{Q_{task}}$, $\mathrm{Est_{Latency}}$, $\mathrm{Est_{Energy}}$)
   ▷ % Compute base stats for scores
2:     $ToGo(tsk) \leftarrow \sum_{i=1}^{N_{acc}} (\sum_{L \in Q_{task}(tsk)} \mathrm{Est_{latency}}(\mathrm{i}, \mathrm{L}))/N_{Acc}$
3:     $Slack(tsk) \leftarrow \mathrm{T_{Deadline}}(tsk) - \mathrm{T_{curr}}$
4:     $\mathrm{T_{queue}}(tsk) \leftarrow \mathrm{T_{curr}} - \mathrm{T_{cmpl}}(tsk)$
5:     $\mathrm{NextLayer}(tsk) \leftarrow Q_{task}(tsk).first$
6:     $PrevAcc(tsk) \leftarrow Stack_{task}(tsk).acc$
   ▷ % Compute Urgency, Preference, and Starvation Scores
7:     $Score_{Urgency}(tsk) \leftarrow ToGo(tsk)/Slack(tsk)$
8:     $Score_{LatPref}(tsk, acc) \leftarrow \frac{\sum_{i=1}^{n} \mathrm{Est_{Latency}}(\mathrm{NextLayer}(tsk), i)}{\mathrm{Est_{Latency}}(\mathrm{NextLayer}(tsk), acc)}$
9:     $Score_{Starv}(tsk) \leftarrow \frac{\mathrm{T_{queue}}(tsk)}{\sum_{\mathrm{i=1}}^{\mathrm{N_{acc}}}(\mathrm{Est_{Latency}}(\mathrm{NextLayer}(tsk), i))/N_{acc}}$
   ▷ % Compute the context-switch cost
10:    $Cost_{switch}(tsk, acc) \leftarrow \frac{\mathrm{CswitchEnergy}(tsk, acc.\mathrm{prevTask}, acc)}{\mathrm{Est_{Energy}}(tsk, acc)}$
   ▷ % Compute Energy Score ($Score_{Energy}(tsk, acc)$)
11:    $Pref_{Energy}(tsk, acc) \leftarrow \frac{\sum_{acc=1}^{N_{acc}} \mathrm{Est_{Energy}}(\mathrm{NextLayer}(tsk), acc)}{\mathrm{Est_{Energy}}(\mathrm{NextLayer}(tsk), acc)}$
12:    $Score_{Energy}(tsk, acc) \leftarrow$
13:        $Pref_{Energy}(tsk, acc) - Cost_{switch}(tsk, acc)$
   ▷ % Compute total MapScore %
14:    MapScore $\leftarrow Score_{Urgency}(tsk) \cdot Score_{LatPref}(tsk, acc)$
15:            $+\alpha \cdot Score_{Starv}(tsk) + \beta \cdot Score_{Energy}(tsk, acc)$
       return MapScore
```

3.1 Urgency Score ($Score_{Urgency}$)

Since we target RTMM ML applications, each inference request has a target deadline defined by the application and input streaming rates (i.e., FPS) from sensors or other data sources. To facilitate scheduling while fully considering such real-time requirements, we propose the urgency score, $Score_{Urgency}$ (Line 7 in Algorithm 1). $Score_{Urgency}$ captures the capability of the underlying accelerators to satisfy the target deadline of an inference task (tsk). It also considers the current system load and progress.

The urgency score is formulated for each inference task, considering the average latency across all accelerators. To define $Score_{Urgency}$ of the tsk, we use $ToGo(tsk)$ and $Slack(tsk)$ described in Algorithm 1 Lines 2-3. $ToGo(tsk)$ quantifies the predicted remaining processing time of a model, and $Slack(tsk)$ quantifies the remaining time until the tsk deadline. Using $ToGo$ and $Slack$, $Score_{Urgency}$ quantifies the ratio of the predicted remaining processing time of a model ($ToGo(tsk)$) and the remaining time until the deadline ($Slack(tsk)$). Based on the equation in Line 7, the urgency score increases either if we need a large amount of time to process an inference or if we have a short amount of available time for the inference request, which effectively models the urgency of the inference.

3.2 Latency Preference Score ($Score_{LatPref}$)

To identify the preference for accelerators for each layer (i.e., which accelerator provides lower latency and energy for each layer), DREAM uses energy and latency estimations generated offline using a cost model or a simulator. We first discuss the preference based on latency in this subsection and discuss how we can utilize energy preference later in Section 3.4.

To obtain the latency preference score, $Score_{LatPref}$ (Algorithm 1 Line 8), we first compute the fraction of latency on a target accelerator (*acc*) to the sum of latency on all accelerators that exist in the system. This quantifies the significance of the latency on an *acc* compared to all other accelerators, which is a lower-is-better metric. By taking the inverse, as shown in Line 8 of Algorithm 1, we obtain $Score_{LatPref}$ as a higher-is-better metric.

3.3 Starvation Score ($Score_{Starv}$)

Due to the workload and hardware heterogeneity, the latency of each layer has a high variance across hardware accelerators. Such diverse latency can lead to the starving of light-weighted layers if the scheduler considers only the time margin to meet the target. For example, if we expect operators A and B to take 1ms and 10ms for processing and both have 12ms until the deadline, a scheduler would schedule layer B first. If such a situation is repeated, heavy-weighted operators can be continuously prioritized, leading to the starving of light-weighted operators. Therefore, DREAM uses a metric to measure the degree of starvation, Starvation score ($Score_{Starv}$). We provide the definition of $Score_{Starv}$ in Line 9 of Algorithm 1.

Starv quantifies the ratio of the wait time (i.e., queue time, T_{queue} in Line 4 of Algorithm 1) and the latency of an operator estimated by a cost model such as MAESTRO [15] and Timeloop [25]. The score increases when a layer waits for a long time to be scheduled. The wait time is divided by the estimated latency to provide a higher *Starv* to light-weighted layers, which are less likely to be scheduled if we only consider the urgency and latency preference.

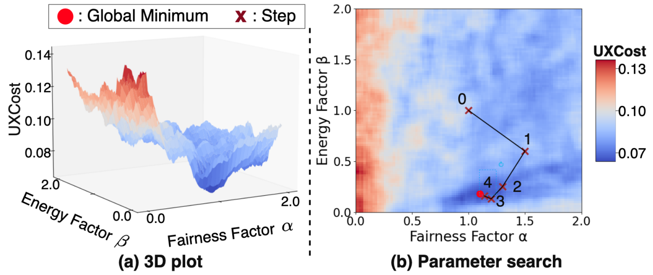
3.4 Energy Score ($Score_{Energy}$)

As many RTMM applications target battery-powered wearable (e.g., AR glasses) or mobile/edge (e.g., drone control) devices, energy is another important factor. DREAM optimizes energy consumption by identifying the most energy-efficient accelerator for each layer with the consideration of the context switch overhead. $Score_{Energy}$ consists of two components: energy preference ($Pref_{Energy}$) and a context-switch cost function ($Cost_{switch}$), as Lines 10-13 in Algorithm 1 show.

Energy preference, $Pref_{Energy}$, is defined in a similar way to $Score_{LatPref}$; compute the significance of energy on an accelerator and take the inverse to obtain a higher-is-better metric. The context switch cost function, $Cost_{switch}$, computes the significance of the extra energy for context switching between two tasks on an accelerator. The extra energy consists of the energy to fetch the activation of a new model from the DRAM and to flush that of the switched-out model to DRAM. We define the energy score, $Score_{Energy}$ as $Pref_{Energy} - Cost_{switch}$ (Algorithm 1 Lines 12-13), which constructs a higher-is-better metric.

3.5 Comprehensive Score: MapScore

Combining all the scoring metrics we discussed, we define a comprehensive metric, MapScore, as defined in Lines 14-15 of Algorithm 1. MapScore utilizes two parameters (α for starvation, β for energy factor), which provide the adaptivity of MapScore to individual systems and usage scenarios with different optimization goals. We use MapScore to drive scheduling decisions in DREAM with α and β optimization methodology we discuss next.



(a) 3D plot

(b) Parameter search

Figure 3: An example search space for scheduling score parameters and optimization steps.

Algorithm 2 UXCost computation

```
▷ Inputs> $Md$(A list of models in the workloads), $T_{exec}$ (A time window where we run workloads)
▷ Output> UXCost ($Md$, $T_{exec}$): UXCost for $Md$ executed for $T_{exec}$
Function Compute_UXCost($Md$, $T_{exec}$)
▷ % Deadline (DL) violation rate and normalized energy for each model
    $Rate_{DLV}[len(Md)] \leftarrow \{0,\}$
    $NormEnergy[len(Md)] \leftarrow \{0,\}$
▷ % $\forall m \in Md$, compute $Rate_{DLV}$ and $NormEnergy$
    for $m$ in $Md$ do
        $NormEnergy[m] \leftarrow \frac{\text{Actual Total Energy Consumption}(m)}{\text{Worst case Energy Consumption}(m)}$
        $Rate_{DLV}[m] \leftarrow \frac{\text{\# DL violated frames}(m, T_{exec})}{\text{\# total frames}(m, T_{exec})}$
▷ % Apply a small number to $Rate_{DLV}$ if no DL violation
        if # DL violated frames($m, T_{exec}$) = 0 then
            $Rate_{DLV}[m] \leftarrow \frac{1}{2 \cdot \text{\# total frames}(m, T_{exec})}$
    end for
    $OverallRate_{DLV} \leftarrow \sum_{m \in Md} Rate_{DLV}[m]$
    $OverallNormEnergy \leftarrow \sum_{m \in Md} NormEnergy[m]$
    UXCost $\leftarrow OverallRate_{DLV} \cdot OverallNormEnergy$
    return UXCost
```

3.6 MapScore Parameter Optimization using UXCost

MapScore serves as a proxy metric during the scheduled time when actual metrics affecting overall user experience (e.g., deadline violation and energy) are unknown. To utilize actual metrics viable to users (deadline violation and energy) for better scheduling decisions, we propose a methodology to provide feedback to MapScore by adjusting starvation and energy factors (α and β). To consider deadline violation and energy together, we construct an energy-delay product (EDP)-like metric, UXCost, as defined in Algorithm 2. Like EDP, UXCost is a lower-is-better metric that equally considers deadline violation rates and energy. However, unlike EDP, UXCost uses deadline violation instead of latency (delay) to reflect the deadline-driven nature of real-time applications. Note

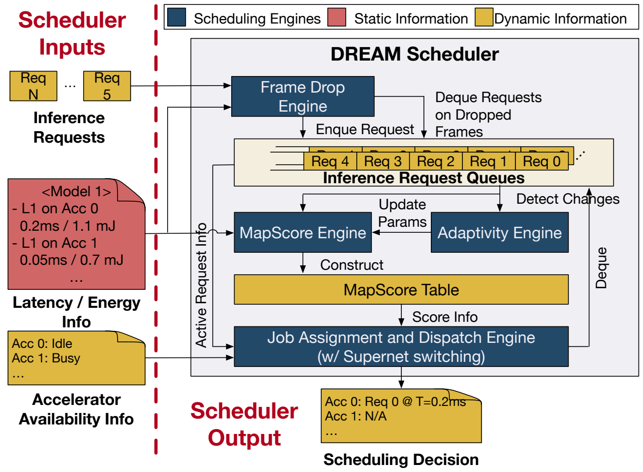


Figure 4: An overview of the structure of DREAM. Yellow boxes represent dynamic information, which is periodically generated (or when an event is detected) while running an RTMM workload. The red box has the latency and energy information for each layer in the models.

that users can modify the formulation of UXCost (e.g., squaring energy) to better align UXCost with user-specific constraints.

Algorithm 2 describes a procedure to compute UXCost, which investigates the deadline violation and energy consumption for each model in a workload. Based on them, the procedure computes the deadline violation rate and normalized energy to the worst case (i.e. energy from worst layer-accelerator pairs) for each model within a given time window, T_{exec} (e.g., 2 seconds). Since the deadline violation rate of zero ($Rate_{DLV} = 0$) leads to UXCost of zero, we apply small numbers based on the number of total frames shown in Algorithm 2 Lines 7-8.

Minimizing UXCost as a goal, we develop an iterative optimization method to optimize parameters (α and β) in MapScore. It samples neighboring and distant parameter pairs, calculates the difference of the minimum two UXCost pairs, and moves to the interpolated point. It repeats this process by decreasing the radius in the next step until the radius is below the threshold. This approach works well in UXCost optimization space over α and β, which are well constrained and well defined, as Figure 3 shows. With such well-conditioned, limited optimization space and quick convergence, we use α and β for providing adaptivity to workload changes, which enhances the overall performance for dynamic workloads. We implement a lightweight online algorithm exploiting the quick convergence of the starvation and energy parameters, α and β. We discuss more detailed results in Section 5.2.

3.7 Why MapScore is a Valid Metric for RTMM

Table 1 lists challenges of RTMM workloads mentioned in Section 2 in columns and highlight which challenge MapScore can handle, compared to other dynamic schedulers for multi-task ML workloads, MoCA [12], Veltair [21] and Planaria [8]. Since all listed dynamic schedulers are deadline-aware dynamic schedulers for multi-accelerators, all schedulers can deal with ML model cascade, ML model concurrency, and real-time timing requirements. However, prior schedulers employ fixed scheduling algorithms for all the usage scenarios, which lead to insufficient adaptivity to the RTMM workload dynamicity. In contrast, DREAM can adapt to various usage scenarios in RTMM applications utilizing the MapScore parameter (α and β) online optimization algorithm we discussed in Section 3.6.

Table 1: The consideration of RTMM challenges in previous schedulers and ours.

Scheduler	RTMM Challenges Consideration					
	Cascade	Concurrent	Real-time	Workload dynamicity		Energy
				Task	Model	
Planaria, Veltair, MoCA	✓	✓	✓			
Ours w/ MapScore w/o param. optimization	✓	✓	✓			✓
Ours w/ MapScore w/ param. optimization	✓	✓	✓	✓	✓	✓

4 DREAM SCHEDULER

Using the scoring metrics we define in Section 3, we implement DREAM scheduler, which handles all the new challenges from RTMM workloads discussed in Section 2.1. We illustrate the overall structure of DREAM in Figure 4 with the internal flow within the scheduler. As inputs, DREAM receives (1) inference requests, which would be periodically generated based on streamed input data, (2) latency and energy information for each layer for each accelerator in the system generated offline using a cost model or a simulator, and (3) accelerator availability information from hardware to determine accelerators that can accept new inference jobs. As output, DREAM generates the scheduling decision, which includes information on the scheduled inference jobs and their target accelerators with the dispatch time.

4.1 Scheduling Flow

DREAM consists of four main software components, labeled as scheduling engines in Figure 4: frame drop engine, MapScore calculator, adaptivity engine, and job assignment / dispatch engine. When an input inference request arrives, the frame drop engine checks the status of the inference request queue to determine acceptance of the inference request. After injecting the request to an inference request queue, the Adaptivity Engine checks changes in workloads and triggers the starving and energy parameter tuning described in Section 3.6 if changes are detected. The MapScore Engine computes the MapScore for requested inference on all the accelerators in the target ML system using the latency and energy information generated from an offline cost model or simulator. The calculated MapScore is stored in the MapScore table. Finally, the job assignment/dispatch engine drives the scheduling decision based on MapScore in the MapScore table, current accelerator availability information (i.e., which accelerator is busy and idle), and current inference requests in the request queue.

The flow includes four core optimizations we implement in DREAM: Light-weighted score metric computation, Light-weighted Adaptivity Engine (with online parameter tuning), Smart Frame Drop, and Supernet Switching.

4.2 Frame Drop Engine

Although many previous works explored frame drop techniques [2, 13, 28] to adjust overall system load to a healthy region, they are not tailored for RTMM. As a result, such works fail to capture key challenges of RTMM workloads: (1) model dependency from cascaded models and (2) task, model, and operator level dynamicity. We clarify key requirements for a frame drop methodology for RTMM workloads.

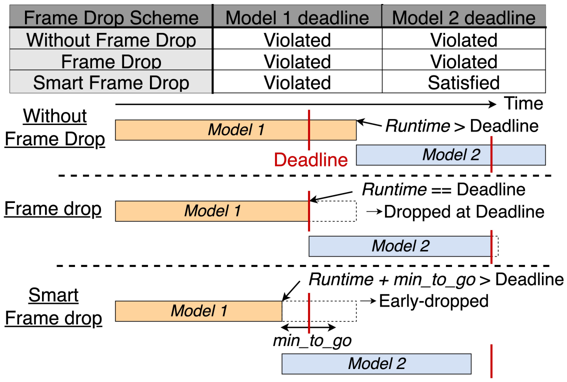
Requirement 1: Considering Model Dependency. When executing cascaded models as an ML pipeline, dropping a frame in a model naturally leads to another frame drop of the next models in the dependency chain. As more than one model can rely on the results of the precedent model, the impact of frame drop on an early model shouldn't be underestimated.

Requirement 2: Exploiting Constrained Dynamicity. Past works rely on some statically defined parameters and mechanisms [2, 13, 28] to handle dynamic workloads, because we cannot predict workloads in the general-purpose computing context. However, on the RTMM workloads, the workloads are not completely random, as all the possible tasks, models, and operators can be identified by the workload specifications (e.g., In a hand-tracking ML pipeline of an AR workload, we know there are only two choices after a hand detection model: not launching or launching a hand tracking model). This opens up RTMM-specific optimization opportunities, which we exploit in the frame drop engine.

4.2.1 Smart Frame Drop Mechanism. Based on the requirements, we develop an RTMM-oriented frame drop mechanism, smart frame drop. In addition to our insights on the RTMM workload dynamicity, Smart frame drop exploits the predictability of latency of each layer in ML accelerators [15, 25] by utilizing the layer-wise latency and energy information input to DREAM as shown in Figure 4. The latency and energy information enables the frame drop engine to estimate the remaining time until the deadlines of a model, which is a part of key information for determining smart frame drop. Figure 5 illustrates our frame drop mechanism, smart frame drop, and compares it against other frame drop mechanisms. Unlike other methods, smart frame exploits RTMM-specific characteristics by identifying the next workloads (worst and best cases based on the constrained dynamicity) and utilizing pre-computed latency information (the predictability of latency for each layer). Using such information, the smart frame drop predicts the possibility of deadline violations before reaching deadlines and proactively drops frames if the deadline is unlikely to be met. As a result, smart frame drop reduces overall deadline violations by providing more time to other models as shown in Figure 5, which is not possible in traditional frame drop methodologies without RTMM-specific workload knowledge.

The frame drop engine is triggered each time a new scheduling decision needs to be made in the job assignment and dispatch engine. The smart frame drop only drops a frame only if all the following four conditions are met.

Condition 1: Deadline Violation Likelihood. Jobs expected to miss deadlines should be the frame drop target. To identify such targets, the smart frame drop mechanism checks the following condition: $minimum\_to\_go > Slack$ where $minimum\_to\_go$ refers to the minimum remaining time until completion, assuming that the best-latency accelerator is used for each layer without any context switching, and *Slack* refers to the remaining time until the deadline.



Frame Drop Scheme	Model 1 deadline	Model 2 deadline
Without Frame Drop	Violated	Violated
Frame Drop	Violated	Violated
Smart Frame Drop	Violated	Satisfied

Figure 5: Comparison of smart frame drop with other frame drop schemes.

Condition 2: Multi-model Violation. More than one active job in accelerators should be expected to violate deadlines. This prevents redundant frame drops when completing the current inference late does not affect other models' deadlines.

Condition 3: Dependency-free. Only the last model in a dependency chain (i.e., an ML pipeline), which does not have any other models that depend on the model, can be a frame drop target. This corresponds to Requirement 1 in Section 4.2.

Condition 4: Maximum Frame Drop Rate. To prevent excessive frame drops on a specific model, the engine bounds the maximum frame drop rate over a specified frame window length. The rate is a configurable parameter, and by default, smart frame drop allows up to 2 drops per 10 frames.

The frame drop engine identifies the frame with the highest $minimum\_to\_go/slack$ among all the frames meeting all four conditions and drops the frame if exists. We interpret the frame drop as deadline violations (completion time = ∞) and consider its impact in UXCost defined in Algorithm 2.

4.3 MapScore Engine

To tackle the dynamicity of RTMM discussed in Section 2.2, we implement a light-weighted online dynamic scheduling mechanism, MapScore Engine, which computes MapScore of the top layers in each inference request queue based on Algorithm 1. MapScore Engine uses pre-computed latency and energy for each accelerator using an offline cost model or simulator. This exploits determinism of latency and energy in accelerators once an inference launches if all data are loaded [15, 25], which is a main difference between dense ML accelerators and other hardware options such as GPUs.

4.4 Adaptivity Engine

When the target workload changes, the optimal starvation and energy factors (α and β) in MapScore also change. Adaptivity Engine, illustrated in Figure 4, detects the workload changes by tracking the inference model list and triggers the starvation and energy factor optimization discussed in Section 3.6. In addition to the workload change scenario, Adaptivity Engine provides adaptivity to system

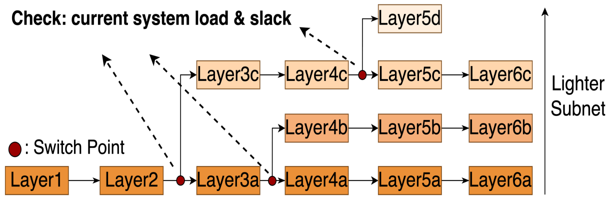


Figure 6: An example of Supernet switching that deploys lighter subnets based on the system load and slack (remaining time until deadline - expected completion time) during execution.

load changes by tracking overall deadline violation rates and frame drop rates, without adding extra latency to the end-to-end latency. It continuously tests a small number of pairs (α, β) around the current value for a short time window and makes a move to a pair that provides the lowest UXCost value. That is, DREAM keeps generating valid schedules while gradually optimizing its internal strategy to a new environment.

4.5 Job Assignment and Dispatch Engine

By default, Job Assignment and Dispatch Engine draw a scheduling decision by selecting the layer-accelerator pair with the highest MapScore. Optionally, Supernet switching can be activated, which uses a technique explored by previous work [4] in the ML algorithm domain.

4.5.1 Supernet-switching. The Supernet weight-sharing technique has been used for training a model once and optimizing the model for different deployment scenarios [4]. Unlike the previous work targeting the static selection of sub-net instances within a Supernet, we explore an RTMM-tailored use case, which is the online sub-net instance switching, as a future-proof form of Supernet switching. This approach helps the system to adaptively decrease the overall system load by deploying lighter model instances, which contributes to better user experiences as it reduces overall deadline violation rates.

Figure 6 shows an example of a Supernet-based model with runtime model instance switching. When the job assignment and dispatch engine generates a scheduling decision on a Supernet layer followed by a switch point, it decides which branch to take by estimating the possibility of whether the current workload can meet the deadline or not. If it cannot meet the deadline, the job assignment and dispatch engine switch to a lighter variant of the Supernet model.

4.5.2 Impact on Accuracy. To prevent accuracy degradation, we carefully select Supernet variants that provide equivalent or greater accuracy than similar-sized or larger networks. For example, Once-for-all [4] variant ofa-s7edge-41 has 96MFLOPs with 73.1% of Top1-accuracy while a fined-tuned MobileNetV3 (at s7edge, similar FLOPs) and MobileNetV2 provides 70.4% and 71.8%-Top1-accuracy, respectively. All the Supernet variants we use (ofa-s7edge) provide higher accuracy with fewer FLOPs than similar-sized models. Note that the online sub-net instance switching implemented in DREAM does not further degrade the model accuracy beyond the baseline static methods [4].

Table 2: Evaluated accelerator hardware settings

Size (# of PE)	Style	Dataflow (PE partitioning)
4K	Homogeneous	2 WS (2K each)
		2 OS (2K each)
	Heterogeneous	1 WS (2K) + 2 OS (1K each)
		1 OS (2K) + 2 WS (1K each)
8K	Homogeneous	2 WS (4K each)
		2 OS (4K each)
	Heterogeneous	1 WS (4K) + 2 OS (2K each)
		1 OS (4K) + 2 WS (2K each)

Table 3: Evaluated Real-time workload scenarios with their FPS targets (FPS) and dependencies (Dep). HD, KS, and FD refer to hand detection, keyword spotting, and face detection models.

Scenario	Application	Model	FPS	Dep.
VR Gaming	Gaze Estimation	FBNet-C [43]	60	
	Hand Detection	SSD_MobileNetV2 [20]	30	
	Pose Estimation	HandPoseNet [22]	30	HD
	Context understanding	Once-for-all [4]	30	
	Keyword Spotting	KWS_res8 [35]	15	
	Translation	GNMT [44]	15	KS
AR Call	Keyword Spotting	KWS_res8 [35]	15	
	Translation	GNMT [44]	15	KS
	Context understanding	SkipNet [42]	30	
Drone (Outdoor)	Object Detection	SSD_MobileNetV2 [20]	30	
	Outdoor Navigation	TrailNet [32]	60	
	Visual Odometry	SOSNet [37]	60	
Drone (Indoor)	Object Detection	SSD_MobileNetV2 [20]	30	
	Indoor Navigation	RAPID_RL [14]	60	
	Obstacle Detection	SOSNet [37]	60	
	Car Classification	GoogLeNet-car [47]	60	
AR Social Interaction	Depth Estimation	FocalLengthDepth [10]	30	
	Action Segmentation	ED-TCN [18]	30	
	Face Detection	SSD_MobileNetV2 [20]	30	
	Face Verification	VGG-VoxCeleb [23]	30	FD
	Context Understanding	Once-for-all [4]	30	

Table 4: DREAM configuration used in evaluation

DREAM Configurations	Dynamic Score Parameter Optimization	Smart Frame Drop	Supernet Switching
DREAM-MapScore	✓		
DREAM-SmartDrop	✓	✓	
DREAM-Full	✓	✓	✓

5 EVALUATION

We evaluate DREAM against three baseline dynamic schedulers, FCFS (first-come-first-served), Veltair [21], and Planaria [8] using five realistic RTMM workload scenarios based on the industry works [17, 32].

5.1 Evaluation Setting

Target hardware. Table 2 lists the hardware systems we evaluated. To show the efficacy of DREAM on various hardware platforms with accelerators, we vary the size of the total number of processing elements (PEs), 4K and 8K, and the dataflows, homogeneous and heterogeneous dataflow with weight-stationary (WS) and output-stationary (OS) dataflows. The WS and OS dataflows are inspired by NVDLA [24] and Shidiannao [7] with further tile size optimizations. We also vary the number of PEs across sub-accelerator instances as shown in Table 2 to model various systems. For all accelerators, we assume 8 MiB of on-chip shared SRAM with 90 GB/s of off-chip bandwidth, running at a 700MHz clock frequency.

Evaluated workload scenarios. We construct five RTMM scenarios, which are shown in Table 3. `AR_Social`, `VR_Gaming` and `AR_Call` are based on XRBench [17], and `Drone_Outdoor` and `Drone _Indoor` are based on TrailMAV [32]. `AR_Social` and `VR_Gaming` have ML pipelines with control and data dependency as "Dep." in Table 3 shows. By default, we activate the dependent workload with 50% of probability. For Supernet switching, we used four Once-for-All [4] model variants for the (visual) context understanding workload. We use four different sub-networks of the Supernet model with weight sharing. We also include operator-level dynamic networks, such as SkipNet [42] and RAPID_RL [14] with early-exit branches. We apply the branch exit probability that each work presented (e.g., the probability of 50% for each block for SkipNet, which reported over 72% of Top-1 accuracy on ImageNet). In addition to dynamicity, the workloads model concurrent ML pipelines. For the outdoor drone scenario, we adopt the workload presented in TrailMAV [32]. For the indoor drone scenario, we replace the navigation model in TrailMAV [32] with RAPID_RL [14], which is tailored for indoor environments. We also use GoogLeNet-car to target in-door parking enforcement use scenarios. For `AR_Social`, we create speaker identifying ML pipeline based on [23] and use its VGG-based model for active speaker verification.

Schedulers. We compare our DREAM with three dynamic scheduling algorithm baselines with different scheduling granularities and strategies:

1. **First-Come-First-Served (FCFS)** [9, 31]: serves the oldest request in the queue immediately if there is a resource available in the granularity of the model.
2. **Veltair** [21]: schedules threshold-based layer-block which groups consecutive layers to prevent scheduling conflicts.
3. **Planaria** [8]: spatially co-locates multiple DNNs by dynamically partitioning the compute resources layerwise based on timing requirements and resource demands.

Note that Veltair is a framework performing both scheduling and compilation that targets a homogeneous CPU cluster. We model their layer-blocking scheme and scheduler. For Planaira, note that it is based on the hardware-software co-design, whereas we model the scheduling component. For simplicity, we refer to the schedulers of Veltair and Planaria as Veltair and Planaria in this section. To analyze the impact of the smart frame drop and Supernet switching, we evaluate the two variants of DREAM along full DREAM, as listed in Table 4. DREAM-MapScore performs MapScore parameter optimization along the score metric-driven job assignments, but without smart frame drop and Supernet switching. DREAM-SmartDrop enables the smart frame drop upon DREAM-MapScore. We set the maximum frame drop rate at 20%. DREAM-Full includes all optimization of DREAM, which indicates DREAM-SmartDrop with Supernet switching.

Evaluation metric. We use UXCost as our main evaluation metric, which is a product of the deadline violation rate and the energy consumption rate. As discussed in Section 3.6, UXCost is a comprehensive metric that considers two key aspects that affect the overall user experience: the deadline violation rate and the energy consumption, which are aligned with the energy-delay product (EDP) used in non-real-time systems.

Latency and Energy Estimation. We use MAESTRO [15] cost model to obtain latency and energy, which reported near 96% accuracy for estimations.

5.2 Results and Discussions

DREAM significantly outperforms baselines. We compare variants of DREAM listed in Table 4 against baseline schedulers in Figure 7 and Figure 8. On average across all workload scenarios and hardware settings, DREAM decreases overall UXCost by 32.1% and 50.0% against Planaria [8] and Veltair [21]. In particular, we observe high improvements for the `AR_Social` scenario with 4K PEs(1WS+2OS), reducing UXCost by 80.8% compared to Planaria, and `Drone_Outdoor` scenario with 4K PEs(1WS+2OS), reducing UXCost by 97.6% compared to Veltair, as shown in Figure 7 (a). For the deadline violation rate decrease, considering heterogeneous hardware by the preference score and heterogeneous workload by the starvation score efficiently handled relatively heavy workloads. For reducing energy consumption, the energy score helped the energy-aware scheduling optimization, while other baselines do not consider energy.

DREAM's HW-heterogeneity-aware scheduling is effective for heterogeneous HW. The overall UXCost gap against DREAM in the heterogeneous environment Figure 7 is larger than in homogeneous results in Figure 8, by 2.20× for Veltair and 1.26× for Planaria. The results imply that a holistic consideration of hardware heterogeneity is important for schedulers; only DREAM considers accelerator size, shape, and dataflow entirely, which distinguishes it from the baselines.

Scheduling is important when the computing power is constrained. When a system has abundant compute resources, the impact of scheduling can diminish as the results in Figure 8 (c). However, as many RTMM workloads are expected to be supported on wearable (e.g., AR glasses) or mobile/edge (e.g., drone) devices with more ML models, constrained compute resources environments are expected to be common cases. For such environments, DREAM demonstrates its strengths. For example, in the 4K 1WS+2OS setting, DREAM on average provides decrease in UXCost by 47.5% and 89.9% compared to Planaria and Veltair, respectively. Unlike baselines, DREAM implements a dynamic scheduling algorithm that utilizes runtime information to adapt to dynamic system load changes, which enables superior results compared to baseline in resource-constrained settings.

DREAM provides considerable energy reduction. Both Veltair and Planaria do not optimize for energy consumption. In contrast, DREAM improves energy consumption even under severe resource constraints based on its adaptive workload adjustment schemes. For example, DREAM decreased energy consumption by 62.9% and 61.4% compared to Planaria and Veltair, respectively, for `AR_Call` on 4K 1OS+2WS system. Although DREAM resulted in a 1.88× and

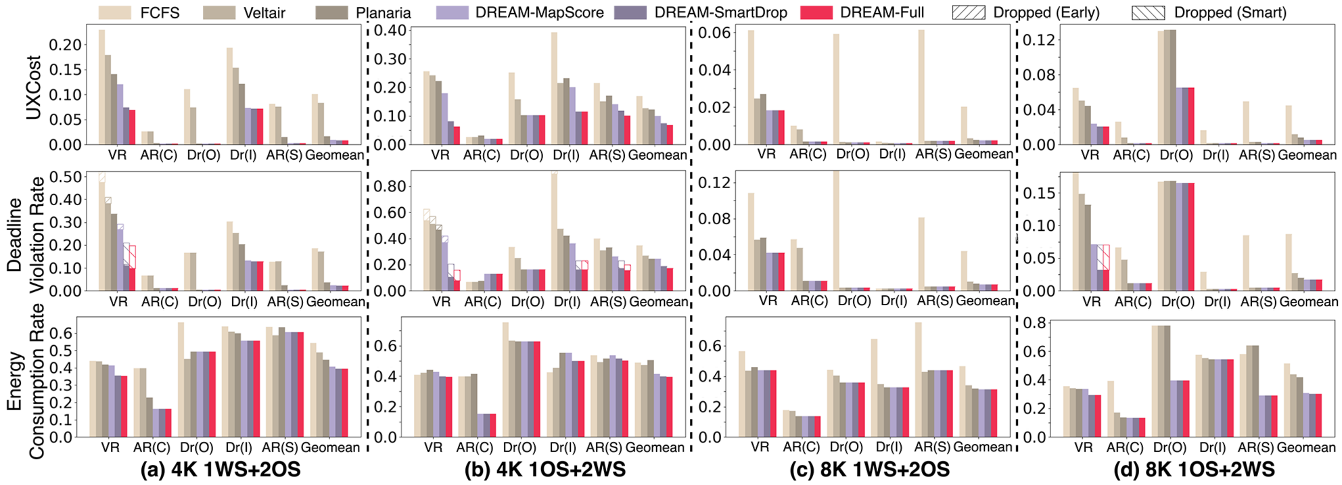


Figure 7: UXCost, Deadline Violation Rate, Energy Consumption evaluation on different workloads and target hardware. Energy consumption is normalized to the maximum possible energy consumption of the target hardware. x-axis denotes workload scenarios.

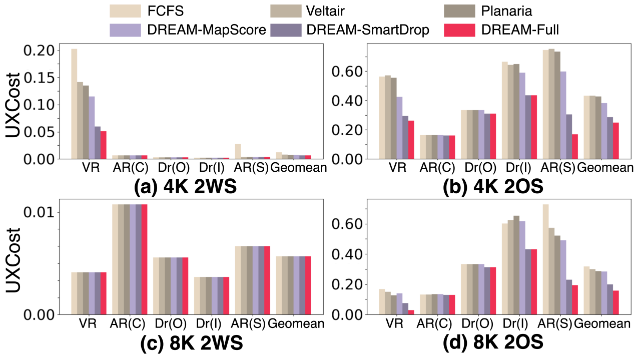
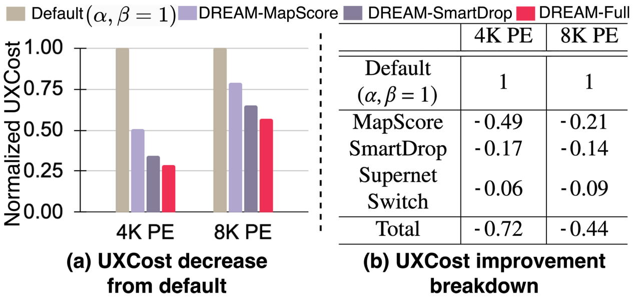


Figure 8: UXCost of homogeneous target hardware settings.

1.66× higher deadline violation rate, overall UXCost decreased by 27.4% and 38.3% respectively, because the baselines are mainly optimized for deadlines. DREAM has the flexibility to explore the balance between deadline and energy by updating the fairness and energy factors (α and β) depending on the application and system requirements. Another example is AR_Social executed on 8K 1OS+2WS where DREAM reduces the energy by 54.6% against the baselines, on average. Such data show that our energy score is an effective way to guide energy optimization in a scheduler.

Each optimization component of DREAM is effective. Figure 9 shows the geometric mean performance improvement breakdown against baseline MapScore with fixed $\alpha, \beta = 1$, as enabling MapScore optimization for MapScore (DREAM-MapScore), enabling smart frame drop (DREAM-SmartDrop), and enabling Supernet switch (DREAM-Full) for VR_Gaming and AR_Social, which contain Supernet models. The result shows the efficacy of all three optimizations. The parameter optimization of MapScore alone shows UXCost decrease by 49.2% for 4K PE and 21.0% for 8K PE case from when MapScore's parameter α, β are fixed to 1. Enabling smart frame drop on top of MapScore optimization shows about 16.5% (4K) and 13.8% (8K). Supernet Switch further decreases UXCost by 6-9% across the configurations.



(a) UXCost decrease from default

	4K PE	8K PE
Default $(\alpha, \beta = 1)$	1	1
MapScore	- 0.49	- 0.21
SmartDrop	- 0.17	- 0.14
Supernet Switch	- 0.06	- 0.09
Total	- 0.72	- 0.44

(b) UXCost improvement breakdown

Figure 9: VR_Gaming and AR_Social geomean UXCost improvement breakdown for each optimization method.

Note that smart frame drop and Supernet switching do not add extra latency because their latency is small and overlaps with the actual execution of workloads. In addition, they do not pose a negative impact on compute resource-sufficient scenarios. For example, Figure 8 (c) shows no difference among DREAM-MapScore, DREAM-SmartDrop, and DREAM-Full, which indicates negligible overhead of those two techniques in compute resource-sufficient scenarios.

DREAM finds near global optimum MapScore parameters under workload changes. Figure 10 shows the MapScore parameter (fairness and energy factors; α and β) search process on four workload change scenarios in the 4K 1OS+2WS setting. For the system booting to application cases (a, b, c), the parameters (α and β) are randomly initialized. Case (d) models a change in the runtime scenario from VR_Gaming to AR_Social, which sets the starting point from the locked parameters in (a). On average across the workload change scenarios, DREAM identified fairness and energy factor pairs that converge within 2% of the global optimum

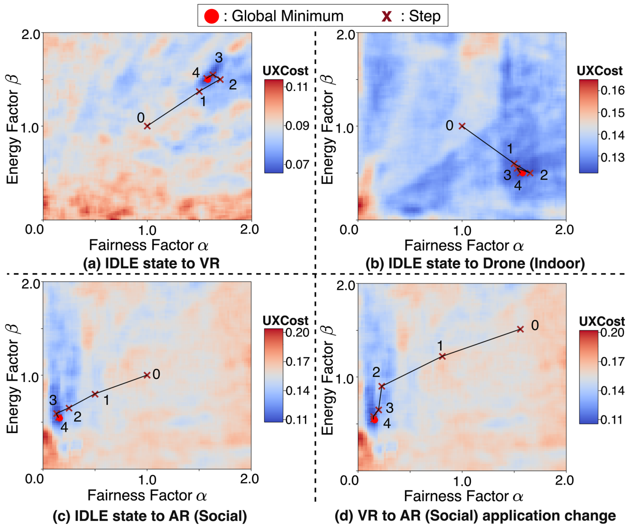


Figure 10: MapScore parameter search. IDLE refers to the state after booting a system with random α and β values.

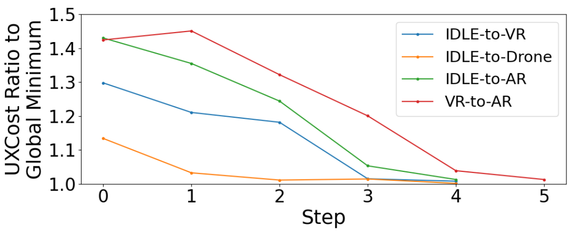


Figure 11: MapScore parameter optimization converge

in UXCost space. The results show that our parameter optimization method successfully reaches a near-global optimum in various workload change cases. This is enabled by the well-conditioned search space with a constrained search range of the parameters within [0,2].

DREAM quickly adapts to workload changes. Figure 11 shows the optimization process steps. For all cases, the optimization progress of MapScore can improve more than 25% of UXCost in just two steps, and within five steps, the parameters of MapScore converge within 2% of the global minimum UXCost. This implies that a system with DREAM is able to quickly adapt to workload changes while processing real-time workloads without significant overhead as the parameter optimization does not block the execution of workloads.

DREAM is effective for dynamic workloads. To evaluate DREAM and baselines in various dynamic workload scenarios, we vary the probability of the ML cascade pipeline of VR_Gaming and AR_Social from 50% to 90%, using 4K heterogeneous accelerators. As Figure 12 shows, DREAM consistently shows better performance than baselines. The improvement is more significant under heavy system load. In particular, DREAM reduces UXCost by 89.8% compared to Veltair and 90.5% compared to Planaria for AR_Social (99%) running in the 1WS+2OS configuration, and by 77.1% and 76.6% for AR_Social (99%) in 1OS+2WS. We also observe that smart frame drop and Supernet switching are effective for dynamic workloads. For instance, for AR_Social (99%) running on the 1WS+2OS configuration, DREAM-SmartDrop reduces UXCost by 48.1% over DREAM-MapScore, and DREAM-Full further reduces it by 65.5%. In the 1OS+2WS configuration, DREAM-SmartDrop reduces UXCost by 53.8% over DREAM-MapScore, and DREAM-Full shows a further reduction of 22.1% compared to DREAM-SmartDrop in 99% probability, whereas 50% probability shows 15.7% and 15.3%, respectively. These results support the efficacy of smart frame drop and Supernet switching for dynamic workloads.

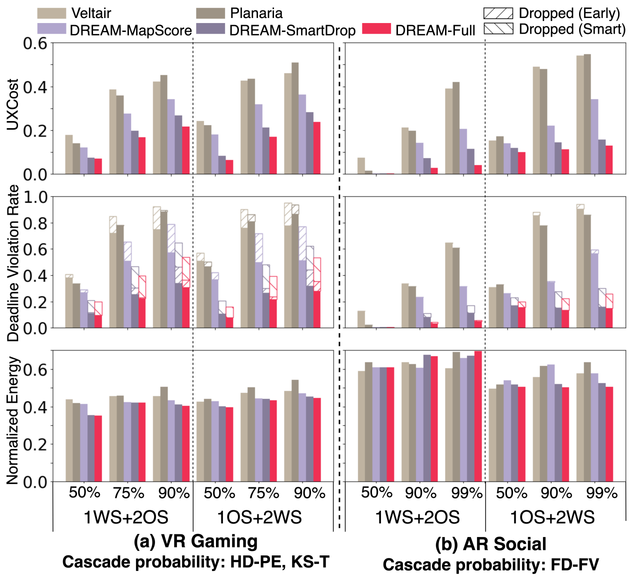


Figure 12: Performance comparison by differing ML cascade pipeline probability.

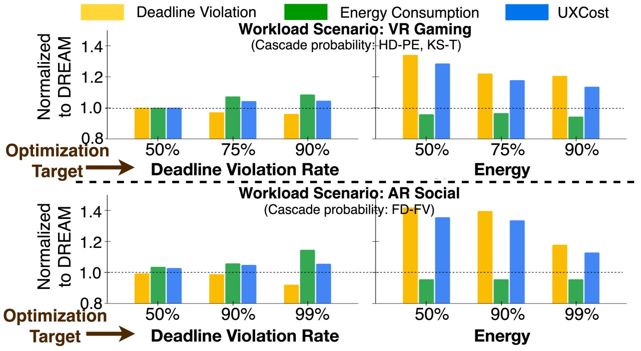


Figure 13: Deadline violation rate- and energy-only optimization results, normalized to DREAM which uses UXCost as an optimization metric. x-axis refers to the cascade probability.

UXCost is an effective metric. Figure 13 shows UXCost, deadline violation, and energy consumption rate result of using deadline violation rate or energy consumption rate as an optimization metric, compared to DREAM which uses UXCost for optimization. Using either one of the metrics may cause undesirable degradation in other metrics. For VR_Gaming, optimizing using deadline violation

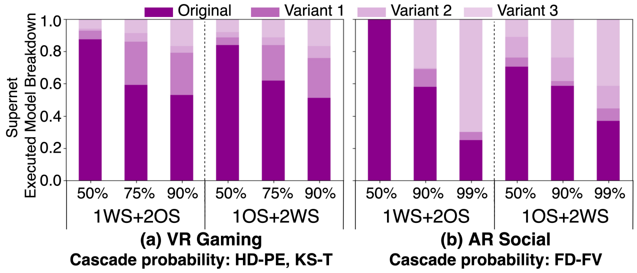


Figure 14: The executed Supernet subnetworks on heterogeneous 4K PE accelerators.

rate can cause an 8.7% increase in Energy consumption, which eventually leads to a 4.6% increase in UXCost. Optimizing Energy consumption would cause a 34.2% increase in deadline violation rate for VR_Gaming with 50% ML cascade pipeline probability, which eventually causes a 28.7% increase in UXCost. For AR_Social, a 14.6% increase in Energy consumption leads to a 5.7% increase in UXCost. Optimizing Energy consumption would cause a 41.9% increase in the deadline violation rate for AR_Social with 50% ML cascade pipeline probability, which eventually causes a 35.7% increase in UXCost. However, UXCost optimization helps balance the benefits between both metrics.

Supernet switching is effective in reducing system load. Figure 14 shows the breakdown of subnets selected for a Supernet for context understanding. "Original" refers to the heaviest subnet (the default). Variants refer to lighter subnets deployed by DREAM's Supernet switching algorithm. Under light system load (i.e., 50% cascade probability), DREAM mainly dispatches the original, more than 80% for VR_Gaming, and 100% for AR_Social on 1WS+2OS. Under heavy system load, we observe that DREAM actively utilizes Supernet switching. For example, more than 40% of the executed Supernet models in VR_Gaming are smaller variants. For AR_Social, more than 60% of Supernet models are lightweight variants. These results show that DREAM successfully detects system load and actively dispatches light-weighted variants to achieve global performance optimization, as Figure 7 and Figure 8 show.

6 RELATED WORKS

We summarize the related works in Table 5 and discuss details of them in three categories as follows:

Schedulers for multi-task DNN accelerators. Prior dynamic schedulers on executing multiple workloads on DNN accelerator, such as Prema [6], propose time-multiplexing DNN execution. However, such temporal co-location of workloads cannot utilize model parallelism for RTMM workloads and often involves preemption overheads. On the other hand, other works proposed spatial co-location of multi-DNN workloads [8, 11, 12, 16, 21]. Herald [16] and MAGMA [11] proposed spatial partitioning of compute or memory resources of the DNN accelerator statically. They both target maximizing the throughput of batched offline workload without latency target, thus not suitable for real-time scenarios with dynamicity. Veltair [21] employs a layer-blocking approach to avoid resource scheduling conflicts on general-purpose CPU clusters. Planaria [8] proposes dynamic allocation of compute resources with a deadline-aware scheduler. MoCA [12] proposes dynamic memory resource partitioning with a deadline and compute-to-memory ratio aware scheduler. However, unlike DREAM, those works do not consider energy and various workload dynamicity.

Table 5: Comparison against prior real-time general-purpose scheduler and scheduler for DNN accelerator.

Target Hardware	Works	Deadline Aware	Heterogeneity Aware	Workload Adaptivity
General Purpose	Deadline-only [1, 3, 5, 26, 27, 33, 46]	✓		
	Harmony [51], *HySARC*^{2} [39], TTSA[50]	✓	✓	
	Skip-over[13], (n,m) [2], (m,k) [28]	✓		✓
	Nexus [31], Clockwork [9]	✓		✓
DNN Accelerator	MoCA [12], Veltair [21], Prema [6]	✓		
	Planaria [8]	✓	✓	
	Herald [16], MAGMA [11]		✓	
	DREAM (This work)	✓	✓	✓

Schedulers for general purpose hardware. Many previous works in the operating system domain explored the scheduling problem on periodic tasks with constraints in real-time system [1, 3, 5, 26, 27, 46]. However, they use offline static scheduling, which is not suitable for RTMM workloads with dynamicity. Joint dynamic and static schedulers [19, 30, 34] have been proposed to reduce the overhead of dynamic scheduling while dealing with an aperiodic task. However, such works target general-purpose systems and do not exploit the predictability of performances of ML workloads on accelerators. Another work [39] proposed algorithms with heterogeneity-aware workload clustering with a deadline-aware earliest-deadline-first algorithm. However, since workloads are pre-clustered before scheduling, this would make the scheduler suffer from resource conflict if workloads with similar computation demands are present. In addition, their scheduler does not consider the dynamic behavior of the workload. Other cloud task schedulers [49, 50] target different optimization goals than this work (e.g. maximizing economic profit of cloud).

Workload management for overloaded system. Many works have proposed various frame drop techniques to enhance the overall system performance under heavy system load. Skip-over [13] employed a static rate-based frame skip technique, but such a static method does not consider model dependency, dynamicity, and other important aspects of RTMM workloads running on accelerators.

Some other works proposed priority-based frame drop methods [2, 28], which determines a subset of tasks to invoke based on off-line priority information. However, focusing on static information is not aligned with the RTMM workload dynamicity characteristics. Its high dynamicity can result in unnecessary or excessive frame drops with the possibility of high cost by forfeiting completed jobs for preceding models in a dependency chain. Some other works such as Nexus [31] and Clockwork [9] target GPU clusters. Nexus [31] proposes to dynamically drop a subset of batches. However, a batch-focused approach is not tailored for RTMM workloads where each inference request mainly consists of a single batch. Clockwork [9] utilizes an FCFS model-wise scheduling method and drops requests upon arrival if the controller predicts that the request will not meet the deadline. However, the FCFS mechanism is not suitable for highly dynamic RTMM workloads, as discussed in our evaluation.

7 CONCLUSION

Emerging RTMM workloads introduce unique challenges to the ML system design, including real-time processing, complex model dependencies and heterogeneity, and various levels of dynamicity. In this work, we clarified the types of dynamicity in RTMM workloads using a taxonomy in various granularities. Based on the RTMM challenges and our insights on the dynamicity, we developed DREAM that holistically considers all the challenges and types of dynamicity in RTMM workloads. In our evaluation using industry-originated RTMM workloads, we observe the importance of such a holistic approach to RTMM workloads codified in DREAM toward low deadline violation rate and energy consumption. Also, we identify that considering global contexts of inference requests on different frames is another key factor, which helped DREAM outperform other state-of-the-art dynamic schedulers [8, 21] on RTMM workloads. Finally, the benefits of Supernet switching we observed in our evaluation indicates that such ML algorithm-system software co-design methods can be future breakthroughs for other similar problems in ML system software.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. We thank Ali Shafiee Ardestani and Krishnakumar Nair for constructive feedback. We thank Harshit Khaitan and Don Stark for their support.

REFERENCES

[1] T.F. Abdelzaher and K.G. Shin. 1999. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems* (1999).

[2] G. Bernat and A. Burns. 1997. Combining (/sub m//sup n/)-hard deadlines and dual priority scheduling. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*. IEEE, San Francisco, CA, USA, 46–57. https://doi.org/10.1109/REAL.1997.641268

[3] A. Burchard, J. Liebeherr, Yingfeng Oh, and S.H. Son. 1995. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. Comput.* (1995).

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations (ICLR 2020)*. https://openreview.net/pdf?id=HylxE1HKwS

[5] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. 2006. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *IEEE International Real-Time Systems Symposium (RTSS 2006)*. IEEE, Rio de Janeiro, Brazil, 101–110. https://doi.org/10.1109/RTSS.2006.10

[6] Yujeong Choi and Minsoo Rhu. 2020. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA 2020)*. IEEE, San Diego, CA, USA, 220–233. https://doi.org/10.1109/HPCA47549.2020.00027

[7] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA 2015)*. IEEE, Portland, OR, USA, 92–104. https://doi.org/10.1145/2749469.2750389

[8] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *Proceedings of The 53rd IEEE/ACM International Symposium on Microarchitecture (MICRO 2020)*. IEEE, Athens, Greece, 681–697. https://doi.org/10.1109/MICRO50266.2020.00062

[9] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Banff, Canada, 443–462. https://doi.org/10.5555/3488766.3488791

[10] Lei He, Guanghui Wang, and Zhanyi Hu. 2018. Learning depth from single images with deep neural network embedding focal length. *IEEE Transactions on Image Processing* 27, 9 (2018), 4676–4689. https://doi.org/10.1109/TIP.2018.2832296

[11] Sheng-Chun Kao and Tushar Krishna. 2022. Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA 2022)*. IEEE, IEEE, Seoul, South Korea, 814–830. https://doi.org/10.1109/HPCA53966.2022.00065

[12] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. 2023. MoCA: Memory-Centric, Adaptive Execution for Multi-Tenant Deep Neural Networks. In *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA 2023)*. IEEE, Montreal, Canada, 828–841. https://doi.org/10.1109/HPCA56546.2023.10071035

[13] Gilad Koren and Dennis Shasha. 1995. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*. IEEE, Pisa, Italy, 110–117. https://doi.org/10.1109/REAL.1995.495201

[14] Adarsh Kumar Kosta, Malik Aqeel Anwar, Priyadarshini Panda, Arijit Raychowdhury, and Kaushik Roy. 2022. RAPID-RL: A Reconfigurable Architecture with Preemptive-Exits for Efficient Deep-Reinforcement Learning. In *Proceedings of the 2022 International Conference on Robotics and Automation (ICRA 2022)*. IEEE, Philadelphia, PA, USA, 7492–7498. https://doi.org/10.1109/ICRA46639.2022.9812320

[15] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2019)*. IEEE, Columbus, OH, USA, 754–768. https://doi.org/10.1145/3352460.3358252

[16] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. 2021. Heterogeneous dataflow accelerators for multi-DNN workloads. In *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA 2021)*. IEEE, Seoul, South Korea, 71–83. https://doi.org/10.1109/HPCA51647.2021.00016

[17] Hyoukjun Kwon, Krishnakumar Nair, Jamin Seo, Jason Yik, Debabrata Mohapatra, Dongyuan Zhan, Jinook Song, Peter Capak, Peizhao Zhang, Peter Vajda, Colby Banbury, Mark Mazumder, Liangzhen Lai, Ashish Sirasao, Tushar Krishna, Harshit Khaitan, Vikas Chandra, and Vijay Janapa Reddi. 2023. XRBench: An Extended Reality (XR) Machine Learning Benchmark Suite for the Metaverse. In *Proceedings of the 5th Machine Learning and Systems Conference (MLSys 2023) (MLSys 2023)*. Miami, FL, USA. https://proceedings.mlsys.org/paper_files/paper/2023/hash/baf570e47e7f4e314a9ffb72c4a5459c-Abstract-mlsys2023.html

[18] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. 2017. Temporal convolutional networks for action segmentation and detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*. Honolulu, HI, USA, 156–165. https://openaccess.thecvf.com/content_cvpr_2017/html/Lea_Temporal_Convolutional_Networks_CVPR_2017_paper.html

[19] J.P. Lehoczky and S. Ramos-Thuel. 1992. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *[1992] Proceedings Real-Time Systems Symposium (RTSS'92)*. IEEE, Phoenix, AZ, USA, 110–123. https://doi.org/10.1109/REAL.1992.242671

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *Proceedings of the 14th European Conference (ECCV 2016)*. Amsterdam, the Netherlands, Portland, OR, USA, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[21] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. 2022. VELTAIR: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022)*. Association for Computing Machinery (ACM), Lausanne, Switzerland, 388–401. https://doi.org/10.1145/3503222.3507752

[22] Meysam Madadi, Sergio Escalera, Xavier Baró, and Jordi Gonzàlez. 2022. End-to-end global to local convolutional neural network learning for hand pose recovery in depth data. *IET Computer Vision* 16, 1 (2022), 50–66. https://doi.org/10.1049/cvi2.12064

[23] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. 2017. VoxCeleb: A Large-Scale Speaker Identification Dataset. In *Procedings of the Interspeech 2017 (Interspeech 2017)*. Stockholm, Sweden, 2616–2620. https://doi.org/10.21437/Interspeech.2017-950

[24] NVIDIA. 2017. NVDLA Deep Learning Accelerator. Retrived from http://nvdla.org.

[25] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS 2019)*. IEEE, Madison, WI, USA, 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[26] K. Ramamritham. 1990. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*

(ICDCS 1990). IEEE, Los Alamitos, CA, USA, 108–109. https://doi.org/10.1109/ICDCS.1990.89256

[27] Krithi Ramamritham. 1995. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems* 6, 4 (1995), 412–420. https://doi.org/10.1109/71.372795

[28] Parameswaran Ramanathan. 1999. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on parallel and distributed systems* 10, 6 (1999), 549–559. https://doi.org/10.1109/71.774906

[29] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. Mlperf inference benchmark. In *Proceddings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA 2020)*. IEEE, IEEE, Valencia, Spain, 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

[30] Ismael Ripoll, Alfons Crespo, and Ana Garcia-Fornes. 1997. An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems. *IEEE Transactions on Software Engineering* 23, 6 (1997), 388–400. https://doi.org/10.1109/32.601081

[31] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*. ACM, Huntsville, Ontario, Canada, 322–337. https://doi.org/10.1145/3341301.3359658

[32] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. 2017. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*. IEEE, IEEE, Vancouver, BC, Canada, 4241–4247. https://doi.org/10.1109/IROS.2017.8206285

[33] John A. Stankovic and Krithi Ramamritham. 1989. The Spring kernel: A new paradigm for real-time operating systems. *ACM SIGOPS Operating Systems Review* 23, 3 (1989), 54–71. https://doi.org/10.1145/71021.71024

[34] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Trans. Comput.* 44, 1 (Jan. 1995), 73–91. https://doi.org/10.1109/12.368008

[35] Raphael Tang and Jimmy Lin. 2018. Deep residual learning for small-footprint keyword spotting. In *Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*. IEEE, IEEE, Calgary, AB, Canada, 5484–5488. https://doi.org/10.1109/ICASSP.2018.8462688

[36] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 23rd international conference on pattern recognition (ICPR 2016)*. IEEE, Cancun, Mexico, 2464–2469. https://doi.org/10.1109/ICPR.2016.7900006

[37] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. 2019. Sosnet: Second order similarity regularization for local descriptor learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019)*. IEEE, Long Beach, CA, USA, 11016–11025. https://doi.org/10.1109/CVPR.2019.01127

[38] Jianming Tong, Yangyu Chen, Yue Pan, Abhimanyu Bambhaniya, Alind Khare, Taekyung Heo, Alexey Tumanov, and Tushar Krishna. 2022. Enabling Real-time DNN Switching via Weight-Sharing. In *The 2nd Architecture, Compiler, and System Support for Multi-model DNN Workloads Workshop (ACSMD 2022)*. New York, NY, USA. https://research.facebook.com/file/703126461319360/enabling-real-time-dnn-switching-via-weight-sharing.pdf

[39] Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Kołodziej. 2015. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems* 51 (Oct. 2015), 61–71. https://doi.org/10.1016/j.future.2014.11.019

[40] Andreas Veit and Serge Belongie. 2018. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV 2018)*. IEEE, Munich, Germany, 3–18. https://doi.org/10.1007/978-3-030-01246-5_1

[41] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. 2021. AlphaNet: Improved Training of Supernets with Alpha-Divergence. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*. PMLR, 10760–10771. https://proceedings.mlr.press/v139/wang21i.html

[42] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV 2018)*. Springer, Munich, Germany, 409–424. https://doi.org/10.1007/978-3-030-01261-8_25

[43] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR 2019)*. IEEE, Long Beach, CA, USA, 10734–10742. https://doi.org/10.1109/CVPR.2019.01099

[44] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. (2016). https://doi.org/10.48550/arXiv.1609.08144 arXiv:arXiv:1609.08144

[45] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR 2018)*. IEEE, Salt Lake City, Utah, 8817–8826. https://doi.org/10.1145/10.1109/CVPR.2018.00919

[46] Jia Xu and David Lorge Parnas. 1990. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on software engineering* 16, 3 (March 1990), 360–369. https://doi.org/10.1109/32.48943

[47] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2015. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR 2015)*. IEEE, Boston, MA, USA, 3973–3981. https://doi.org/10.1109/CVPR.2015.7299023

[48] Haoran You, Cheng Wan, Yang Zhao, Zhongzhi Yu, Yonggan Fu, Jiayi Yuan, Shang Wu, Shunyao Zhang, Yongan Zhang, Chaojian Li, et al. 2022. EyeCoD: eye tracking system acceleration via flatcam-based algorithm & accelerator co-design. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA 2022)*. ACM, New York, NY, USA, 610–622. https://doi.org/10.1145/3470496.3527443

[49] Haitao Yuan, Jing Bi, Wei Tan, and Bo Hu Li. 2016. Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds. *IEEE Transactions on Automation Science and Engineering* 14, 1 (Feb. 2016), 337–348. https://doi.org/10.1109/TASE.2016.2526781

[50] Haitao Yuan, Jing Bi, Wei Tan, MengChu Zhou, Bo Hu Li, and Jianqiang Li. 2016. TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds. *IEEE transactions on cybernetics* 47, 11 (July 2016), 3658–3668. https://doi.org/10.1109/TCYB.2016.2574766

[51] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. 2014. Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE transactions on cloud computing* 2, 1 (2014), 14–28. https://doi.org/10.1109/TCC.2014.2306427