Interference-Aware DNN Serving on Heterogeneous Processors in Edge Systems

Yeonjae Kim*, Igjae Kim*, Kwanghoon Choi*, Jeongseob Ahn†, Jongse Park*, Jaehyuk Huh*
*School of Computing, KAIST, Daejeon, South Korea
†School of Electrical Engineering, Korea University, Seoul, South Korea
{yjkim, ijkim, khchoi}@casys.kaist.ac.kr, jsahn@korea.ac.kr, jspark@casys.kaist.ac.kr, jhhuh@kaist.ac.kr

***Abstract*—With growing demands on the acceleration of machine learning (ML) computation, processors for edge systems have been integrating heterogeneous devices such as GPUs and NPUs (Neural Processing Units) for ML computing. However, as multiple ML models need to be processed simultaneously even in edge systems, ML inference schedulers for such heterogeneous devices must not only consider the efficiency of devices for different ML models, but also consider the interference among them carefully. Based on our analysis on the behaviors of inter-device interference, the study first builds an interference prediction scheme using a multilayer perceptron model. The interference prediction model is trained with the data from randomly generated ML models, and thus it can be prepared without any prior knowledge of actual target ML workloads. Using the highly accurate prediction model, this study proposes a goal-independent scheduling framework, which allows any scheduling objective set by users. The scheduling framework uses a sampled simulation method to support such flexible scheduling goals with a minimized latency. Our experimental results on a commercial edge system show that our framework backed by the interference prediction model can effectively improve performance for diverse goals.**

I. Introduction

The widening adoption of machine learning (ML) for many critical computing problems has been accelerating the integration of GPU and NPUs (Neural Processing Units) in processors targeted for mobile and edge platforms. In such heterogeneous processors, scheduling jobs to different computing devices poses new challenges. Recent studies proposed scheduling techniques to optimize the mapping of ML tasks and heterogeneous computing devices for certain fixed goals [1], [2].

However, scheduling in edge systems with heterogeneous processors must consider two important requirements. First, the execution characteristic of an ML task is not only determined by the computing device it is running on, but is also highly affected by the co-running tasks in other devices. Second, another important requirement for edge schedulers is the configurability of scheduling policies. Users have diverse scheduling objectives, and the scheduler design must be able to support such flexibility.

To address the challenge, this paper first investigates the intensity and characteristics of interference among co-located ML tasks for heterogeneous edge processors. Our analysis shows that the interference can often increase the latency of an ML task significantly, disrupting the expectation assumed by schedulers. Based on the analysis, this study proposes an MLP (Multilayer Perceptron) model to predict interference. By training the interference model with randomly generated data, the interference model can be trained without any prior knowledge of the target real ML models. Our results show that the interference model has a high accuracy of 95.9%, when it is validated with 14 real ML tasks.

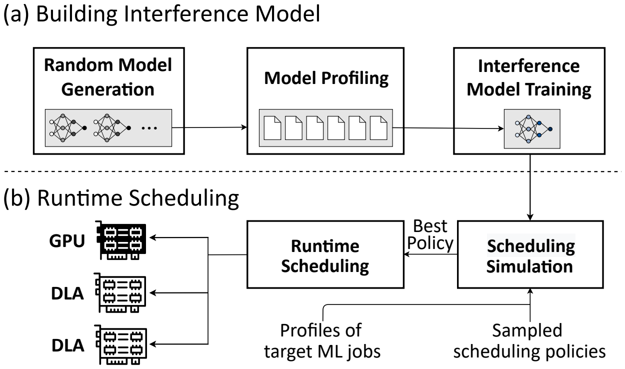


Fig. 1: Interference model and runtime scheduling.

Incorporating the interference model, we propose a goal-independent scheduling framework based on sampled simulation [3] to reduce simulation space. It simulates the results of possible sampled scheduling options and chooses one satisfying the given objective. Unlike the prior goal-specific schedulers, the benefit of the approach is that it can support any goal. During the simulation, the interference model is used to accurately predict the execution latency of ML tasks on GPU or NPU. Figure 1 shows the overall design of the interference model construction and runtime scheduling. The construction of the interference model is needed once for a platform, since it is not tied to any specific ML tasks running on the platform.

We implemented the scheduling framework with the new interference prediction model on the NVIDIA Xavier platform [4]. Our evaluation with 14 real world ML workloads shows that compared to the baseline policy, it can improve goodput (throughput satisfying the latency SLO) by 40% and 99% SLO throughput by 36.1% on average. The results show that the interference model is critical to meet the given scheduling objective, and our goal-independent scheduling framework can support diverse goals effectively.

II. Background

A. DNN Scheduling on Heterogeneous Edge Processors

Recently, a series of works [1], [5] investigated the scheduling problem for heterogeneous processors. Gavel proposes a scheduling mechanism for DNN training on multigenerational GPUs by formulating the scheduling as an optimization problem [5]. However, it targets batch-oriented training tasks, but our work is for real-time inference scheduling which must consider the latency of each individual inference request. PSLO-MAEL proposes DNN inference scheduling algorithms on heterogeneous processors in edge systems through estimating expected latency and leveraging model slicing [1]. The prior works proposed profiling-based inference scheduling mechanisms based on statically profiled execution statistics. However, these profiling-based methods lack consideration of the interference among the co-running inference tasks.

In contrary, this work focuses on the edge systems that have heterogeneous processors and a unified memory shared by the processors, and studies the interference implications. In addition, our scheduling framework provides a customizable objective function by using simulation-based scheduling.

B. Modeling Interference

Performance interference among co-running applications has been causing unpredictable behaviors in multi-core systems. To investigate the interference effects of co-located applications in multi-core systems, Bubble-Up profiled the application-specific interference sensitivity [6]. Bubble-Up produces stress to the memory subsystem, called *bubble*, while incrementally enlarging the bubble to obtain the sensitivity curve. Mage is an interference-aware runtime system, which optimizes performance and efficiency for co-located applications in distributed and multi-core systems [7]. Gpulet devises a novel scheduling framework for multi-GPU servers that incorporates both spatial and temporal sharing while considering interference [8]. Other prior work introduced an interference modeling approach, which predicts the performance degradation due to consolidation [9], [10] .

III. Motivation

A. Contention on Shared Resource

This study identifies two primary types of shared resources, which largely impact the inference latency: 1) memory bandwidth, and 2) shared use of GPU execution resources.

1) Contention on Memory Bandwidth: CPU, GPU, and NPU in an SoC compete over the memory controllers, producing different demands of bandwidth usage. For instance, since the NVIDIA Xavier system has a single shared external memory controller, inference executions on DLAs could be bottlenecked by the memory-intensive inference jobs on GPU. **Methodology:** We pin a specific DNN model to a processor, while running two other DNN models on the rest of the processors. We run two experiments: 1) ShuffleNet pinned to GPU, and 2) GoogLeNet pinned to DLA. For each experiment, we populate all possible pairs of benchmark DNN models and place the pairs with the pinned model. Depending on which combinations of models are co-running, their memory bandwidth utilization varies so we use the model combinations as a knob to navigate different memory bandwidth utilizations. **Implication of memory bandwidth contention:** Figure 2 illustrates the inference execution time of ShuffleNet (GPU) and GoogLeNet (DLA) normalized to their respective sole execution baseline. We observe a trend that there is a correlation between 1) memory bandwidth contention of co-located tasks at the memory controller, and 2) the performance degradation on the inference task. However, note that while the trend is observable, the correlation is not always very strong, which implies that there are other factors to interplay with the bandwidth utilization to determine the interference effects.

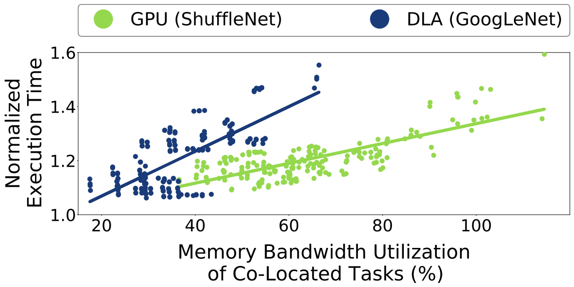


Fig. 2: Normalized execution time of ShuffleNet (GPU) and GoogLeNet (DLA) when other two DNN models are co-running on the rest of the processors.

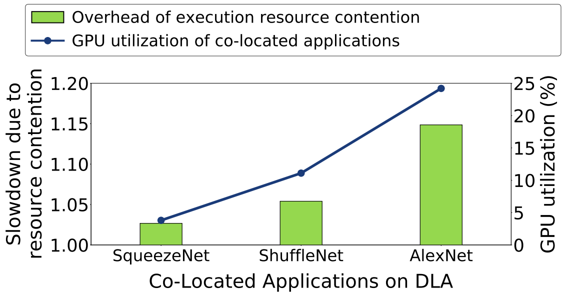


Fig. 3: Slowdown due to resource contention of ResNet-50 (GPU) when DNN model is co-running on DLA.

2) Contention on GPU Execution Resource: GPU execution unites are considered as source of resource contention. **Contention on GPU execution resources:** First, NPUs might have insufficient capabilities to support diverse layers in DNN. When NPUs does not support for a specific layer, they often rely on GPUs for processing. Seconds, NPUs are often equipped with insufficient on-chip memory resources. In such cases, there may be a fallback to a GPU when a NPU lacks enough on-chip memory space to hold the entire working set. **Methodology:** We pin ResNet-50 on GPU, while running one of the three DNN models, SqueezeNet, ShuffleNet, and AlexNet, on the DLA. To exclusively quantify the interference effect by execution resource contention, we measure the sum of latencies of execution on the GPU with ResNet-50 in an interfered environment and compare it with the end-to-end latency. In this way, it allows us to calculate the isolated

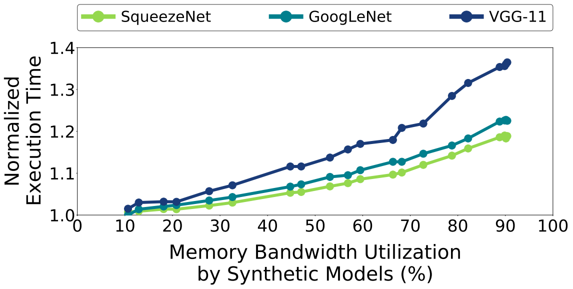


Fig. 4: Sensitivities to contention on memory bandwidth utilization of three different DNN models running on DLA.

effect of execution resource contention, where the bandwidth contention does not contribute to the latency measurement.

Quantification of contention: Figure 3 reports the normalized execution time of ResNet-50, when interfered with one of the three DNN models along with their corresponding GPU execution unit utilizations. The results show that as the co-running model changes, the performance degradation due to resource contention varies significantly, ranged from 3% (SqueezeNet) to 15% (AlexNet). As AlexNet consumes the largest GPU utilization (24.2%), it incurs the greatest performance degradation, while SqueezeNet with the least utilization (3.8%) produces the smallest performance loss. The results also show that the contention on execution units contributes to a significant portion of total performance degradation.

B. Sensitivity on Interference

Another factor that impacts on the interference of heterogeneous processors is *sensitivity on interference.* This metric is introduced in a prior work, Bubble-Up [6], which studied the interference of multiple cores and proposed an inference model. Different DNN models have different sensitivity to the resource contentions and therefore experience different levels of performance degradation when the contentions emerge. Thus, this model-specific property is an important factor to consider in the interference prediction model.

Methodology: We use a similar methodology as in Section III-A with a few modification. We pin SqueezeNet, GoogLeNet, and VGG-11 to DLA, while executing a *synthetic DNN model* on GPU. This synthetic model consists of a single convolution layer. We modify the number of output channels of this layer to regulate its memory traffic. The synthetic model works as *bubble* as in the prior CPU-based interference study, Bubble-Up [6]. In this way, we precisely control the memory bandwidth of synthetic model, which in turn facilitates the interference analysis for the three DNN models.

Different interference sensitivities of different DNN models: Figure 4 shows the normalized inference execution latency using each model's solo execution latency as baseline. We observe that VGG-11 is significantly more sensitive to the availability of memory bandwidth since it requires not only to read large weight parameters from off-chip memory, but also to write large output tensor to the memory. In contrast, SqueezeNet and GoogLeNet are lightweight models in terms of weight parameters, which incur less pressure to the memory controller, and therefore, they are less sensitive to the memory bandwidth utilization.

IV. Prediction Method for Interference

A. Learning to Predict Interference

In the previous section, we investigated two sources of interference, (1) the memory bandwidth and (2) GPU execution unit. By focusing on the two main sources, the interference model is designed to predict the latency increase by co-runners. Given that our target system contains one GPU and two DLAs, the prediction model consists of four sub-models:

- Model 1 (GPU ← DLA) predicts the latency degradation of a GPU task, when a co-runner is running in a DLA.
- Model 2 (DLA ← GPU) predicts the latency degradation of a DLA task, when a co-runner is running in the GPU.
- Model 3 (GPU ← DLA0, DLA1) predicts the latency degradation of a GPU task, when two co-runners are running in the two DLAs.
- Model 4 (DLA0 ← DLA1, GPU) predicts the latency degradation of a DLA task, when a co-runner runs in the GPU, and the other co-runner runs in the other DLA.

We initially attempted to build an interference estimation method by understanding how the architecture resources are shared across different inference tasks. Such an empirical heuristic, however, failed to provide sufficient results to be used for diverse ML inference tasks at runtime. This is because the interference effect is a consequence of complex interactions between various ML tasks, making it impractical to devise a unified model that precisely quantify the performance interference.

To this end, inspired by prior works that leverage machine learning (ML) to solve system problems [11], we employed lightweight MLP models, noting their predictions were sufficiently accurate.

B. Modeling Interference

The key design goal of our interference prediction scheme is to create a workload-independent model. Each sub-model employs a multilayer perceptron(MLP) architecture, and the network architectures are shown in the last row of Table I. The input features for the interference model include memory bandwidth of all running tasks, the GPU utilization of DLA tasks, the average layer execution time of GPU task, and sensitivity values of the target task. Since the GPU utilizations of most GPU tasks are almost same, we exclude GPU utilization of GPU tasks as input features. For model 4, to improve prediction accuracy, we used an additional sensitivity value.

Sensitivity refers to the latency degradation of a target task when one or two co-running tasks are executed on the GPU or DLA. The sensitivity is measured using the method discussed in Section 3.3. When the target ML model runs onm the GPU or DLA, a fixed synthetic model is executed on the other device to generate interference pressures. Unlike Bubble in the prior work [6], we use a single sensitivity value from a

		Model 1	Model 2	Model 3	Model 4
Interfered Processor		GPU	DLA	GPU	DLA0
Co-running Processor		DLA	GPU	DLA0, DLA1	GPU, DLA1
Input Features	Memory Bandwidth Utilization	G_APP, D_APP	G_APP, D_APP	G_APP, D0_APP, D1_APP	G_APP, D0_APP, D1_APP
	GPU Utilization	D_APP	D_APP	D0_APP, D1_APP	D0_APP, D1_APP
	Average Layer Execution Time	G_APP	G_APP	G_APP	G_APP
	Sensitivity for (GPU, DLA)	G_APP	D_APP	-	D0_APP
	Sensitivity for (GPU, DLA0, DLA1)	-	-	G_APP	D0_APP
Network Architecture		5-64-256-256-1	5-128-16-1	7-256-256-64-1	8-256-128-32-1

[*] G_APP and D(0/1)_APP denote whether the input feature is measured with the application in the GPU or DLA (0/1).

TABLE I: Input features of the interference model consisting of 4 sub-models

Configuration		
Block Type		Sequential, Add, Concat
Sequential Type		Conv+(BN)+ReLU, FC+(BN)+ReLU, Conv+(BN)+ReLU+Pool
Conv	Output Channels	3 - 1024
	Kernel Size	1 - 11
	Padding	0 - 11
	Stride	1 - 11
FC	Output Features	32 - 4096
	Activation	ReLU, SoftMax
Pool	Type	MaxPool, AvgPool
	Padding	0 - 2
	Kernel Size	1 - 7
	Stride	1 - 7

TABLE II: Random model generation: specification for random models for training the interference model

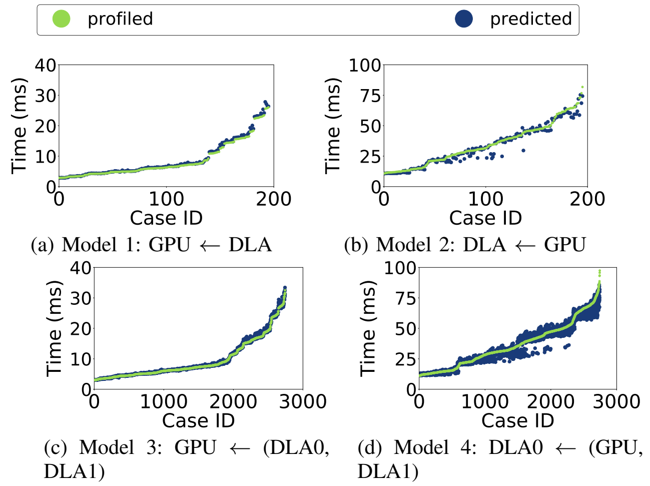
fixed pressure. For our purpose, this simplification produce a model that is sufficiently accurate.

For the workload-independent model, the training data for each MLP are generated from randomly generated ML models, and the features are collected from the layers of these models by executing them on the target system. By using the collected data from the random ML tasks, the interference model is prepared without any prior knowledge of the actual target ML workloads. This goal is necessary to support future additions of new ML models, without retraining the interference model.

C. Training the Interference Model

Training the interference model is necessary only once for a platform, unless there is a significant change in the hardware configuration or software stack that impacts performance. Once the prediction model is built, it does not need to be retrained even if the platform needs to serve new ML models.

Random DNN generation: To allow building workload-independent models, we generate random ML architectures and profile the input features from the execution of the random ML models. We use 155 random computational graphs of DNN models, as used by the prior study [12]. Table II shows the parameters for random DNN model generation. For the semantically meaningful generation of DNNs, a random computational graph is the connection of blocks. A block reflects common topology in DNNs, and it can be a sequential block, add block, or a concatenation block. The range of parameters for convolutional, fully-connected, and pooling layers are also shown in the table. Once the random DNN models are generated, they are mapped to the hardware platform, and their input features and interfered latencies are measured. The collected data are then used for model training.



(a) Model 1: GPU ← DLA
(b) Model 2: DLA ← GPU
(c) Model 3: GPU ← (DLA0, DLA1)
(d) Model 4: DLA0 ← (GPU, DLA1)

Fig. 5: Accuracy of the predicted time over the measured (profiled) time

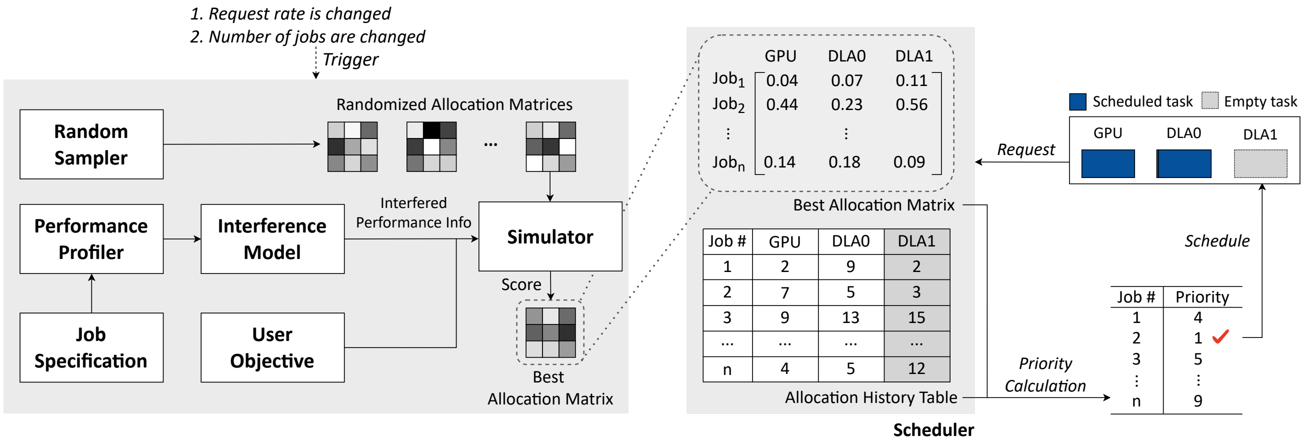
D. Interference Prediction

To predict interference using the trained model, each target ML task is profiled by running it in a GPU and DLA. The profiled input features of both the target and co-runner tasks are fed into the interference model to predict the latency degradation to the target ML task.

Interference prediction accuracy: The prediction model is trained from the randomly generated models as discussed in Section 4.2. We run 14 DNN models on both a GPU and DLAs with all combinations, and validate the measured results match the predicted ones. Table III shows the details of 14 DNN models.

In figure 5 presents the execution time plots for the actual runs and predicted ones. In figures 5 (a) and (b), one DNN model runs on the GPU, while another DNN runs on one of the DLAs. In figures 5 (c) and (d), one DNN runs on the GPU and two DNN models run on two DLAs. The left two figures (a) and (c) show the execution times for the DNN model on the GPU, while the right two figures show execution times on DLAs.

As shown in the figure, the prediction model demostrates a very high accuracy. For (a) and (b), the average accuracies are 97.8% and 94.3% respectively. For (c) and (d), the average accuracies are 97.4% and 94.0% respectively. These results show that our prediction model can produce a highly accurate estimation of interference for both GPU and DLA execution, although it is trained with randomly generated ML models.



$$\begin{array}{cccc} & GPU & DLA0 & DLA1 \\ Job_1 & 0.04 & 0.07 & 0.11 \\ Job_2 & 0.44 & 0.23 & 0.56 \\ \vdots & & \vdots & \\ Job_n & 0.14 & 0.18 & 0.09 \end{array}$$

Job #	GPU	DLA0	DLA1
1	2	9	2
2	7	5	3
3	9	13	15
...	...	...	...
n	4	5	12

Job #	Priority
1	4
2	1 ✓
3	5
⋮	⋮
n	9

(a) Sampled Simulation

(b) Runtime Scheduling

Fig. 6: Scheduling system overview: (a) sampled simulation and (b) runtime scheduling

V. Scheduling System

A. Overview

This section presents our scheduling framework based on sampled simulation. The runtime component of the scheduler routes requests to different devices based on the priority score for each device. The priority score of a request is determined by how much each device is allocated for the ML model by scheduling and how much credit it has already consumed in the current scheduling cycle. To find the best allocation for each ML model, we use a simulation method that simulates the scheduling and execution of requests to estimates the outcome. To reduce the search space of the priority score, candidate allocation samples are randomly generated and evaluated with the simulation method. If a sampled resource allocation passes the scheduling objective, it is selected and used for actual scheduling. Figure 6 illustrates the sampled simulation (a) and runtime scheduling (b).

Configurable objective: One of the key features of our scheduling mechanism is its support for configurable objectives. To achieve this, the scheduling mechanism must be flexible enough to accommodate various goals. The scheduling policy is represented by an *allocation matrix*, which specifies the ratios for using each device for current ML workloads. To support a variable policy, we use a generic simulation-based method. By evaluating possible options through fast simulation, this method finds the right allocation matrix which satisfies the policy.

B. Priority-based runtime mechanism

The runtime component of the scheduler routes each request based on the priority score assigned to each device. When an empty slot becomes available on one of the computing devices, the runtime system selects the request with the highest priority for that devices. The priority score of a request depends on two factors, *allocation ratio* and *consumed allocation*. For each request, the allocation ratio for a device represents the proportion of device time the request is entitled to use during a scheduling period. *Allocation matrix* contains the allocation ratios for all workloads across the computing devices in the system.

The best allocation matrix, produced by the sampled simulation, contains the chosen allocation ratio for each device, as shown in Figure 6 (b). Additionally, the scheduler maintains *allocation history table*, which records the number of time slots each ML model has consumed during the current period. The allocation history table is reset periodically to replenish the credit available for using the devices. The actual priority is defined as follows:

$$Priority\ Score = \frac{Allocation\ Ratio}{Consumed\ Allocation}$$

The priority is high when the initial allocation is large and remains unused in the current scheduling cycle. The priority-based mechanism is work-conserving. If a GPU or DLAs becomes idle, the next request with the highest score is routed to the idle device, even if the priority of the job for the device is lower.

C. Goal-independent Simulation

To generate the best allocation matrix, we use a simulation-based method to allow customizable goals. This method simulates the scheduling and execution of jobs based on their request rates and a given allocation matrix. Figure 6 (a) shows the simulation method. As shown in the figure, a random sampler generates possible allocation matrices. The number of random samples should be large enough to include at least one good scheduling which satisfy the scheduling goals.

For each random allocation matrix, the simulator mimics the scheduling and execution with the latency of each request on GPU or DLA. Requests are generated with specified rates and are mapped to virtual GPU and DLAs according to the current allocation matrix. Execution times are estimated with the profiled latency and interference prediction model. Virtual GPU and DLAs times advance by the estimated execution times, and the scheduler assigns new requests on them. After

Name	# Params	On GPU			On DLA			Speedup[4]
		MEM BW[1]	GPU UTIL[2]	LAT[3] (ms)	MEM BW	GPU UTIL	LAT (ms)	
AlexNet	61M	31.6%	96.7%	4.0	19.9%	24.2%	10.2	2.6x
ResNet-18	12M	31.5%	97.7%	5.8	18.5%	1.5%	24.3	4.2x
ResNet-34	22M	27.8%	98.0%	10.5	19.8%	0.9%	40.5	3.9x
ResNet-50	26M	45.8%	98.0%	15.0	21.7%	0.9%	52.9	3.5x
SqueezeNet 1.0	1.2M	41.1%	97.6%	6.3	14.7%	2.6%	19.7	3.1x
SqueezeNet 1.1	1.2M	46.1%	95.8%	3.1	14.5%	3.8%	13.3	4.3x
Inception v3	27M	66.3%	98.2%	21.3	32.7%	25.2%	51.5	2.4x
GoogLeNet	13M	81.3%	98.0%	13.5	33.2%	19.8%	35.8	2.7x
ShuffleNet V2 x1.0	2.3M	39.0%	91.9%	2.7	8.7%	11.1%	16.0	5.9x
ShuffleNet V2 x0.5	1.4M	46.9%	94.9%	4.5	13.6%	17.5%	19.0	4.2x
MobileNet V2	3.5M	57.7%	96.6%	5.7	14.8%	1.2%	42.6	7.5x
MNASNet 1.3	6.3M	45.5%	96.5%	4.7	8.8%	3.4%	28.6	6.1x
MNASNet 1.0	4.4M	42.3%	97.2%	6.9	14.7%	9.3%	33.4	4.8x
MNASNet 0.5	2.2M	33.0%	97.0%	4	21.0%	21.4%	10.1	2.6x

[1] Memory Bandwidth Utilization
[2] GPU Utilization
[3] Latency
[4] Runtime Speedup of GPU over DLA

TABLE III: 14 benchmark ML models

simulating all requests, the outcomes are evaluated. The expected result will vary based on the scheduling objective. Once the simulation of all generated allocation matrices is complete, the best allocation matrix is selected according to the specified scheduling goal.

Simulation for scheduling is necessary only when the current workload changes. If a new ML model is added or an existing ML model is removed, the scheduling simulation is initiated. Significant changes in requests rate can also trigger a simulation to adjust scheduling, as demonstrated in prior work [13].

D. Sampled Allocation Matrices

One downside of the simulation method is its extensive search space. As the number of ML models increases, possible allocation matrices increase exponentially. Although the simulation of one matrix takes a very short time in our platform, the total simulation time required for an exhaustive evaluation become impractical for scheduling.

To make the problem solvable, we adopt a sample-based evaluation which is based on the findings in the prior work [3]. The prior statistical approach analyzed that a small number of samples can include a scheduling that is within a certain bound from the best one found through exhaustive evaluation. The study showed that hundreds or even a thousand of samples can provide a good scheduling with less than 1% performance difference from the best one, even when the total number of possible schedules is in the millions or 10s of millions. Using the sampled approach, we generate 1000 random allocation matrices to identify a good allocation candidate satisfying the goal. Simulating these 1000 random allocation matrices takes 200 milliseconds on the CPU of the Xavier platform. Since the simulation is needed infrequently due to changes in workloads and request rates, and it consumes only CPU resources, the simulation latency does not significantly impact the overall performance.

VI. Evaluation

A. Methodology

We implemented our scheduling framework with the interference prediction model on the NVIDIA Jetson AGX Xavier system [4]. The processor has an 8-core CPU, one GPU, and two DLAs for ML acceleration. Our framework is built on top of the NVIDIA TensorRT [14]. Once scheduling decision is made, the request is assigned to GPU or DLAs using TensorRT APIs. For scheduling on the platform, there is a restriction for context switching. For two DLAs, up to 4 active contexts are supported, while the GPU does not have such a restriction.

Benchmark: We use 14 DNN models from the torchvision for the evaluation [15]. Table III presents the characteristics of the models when they are running on a GPU or DLA. The last column of the table shows the speedup of running each model on a GPU over a DLA, which ranges from 2.4x to 7.5x. The GPU exhibits significantly lower latencies across the ML tasks.

Using the 14 DNN models,we construct 40 application scenarios (8 sets × 5 request ratios). Table IV presents the DNN mixes and ratios. The 40 scenarios are divided into 8 sets. Each set uses five different request ratios for ML models in that set. Sets 1-4 run three different DNN models and Sets 5-8 run six different DNN models. The last column shows the range of interference observed in each set.

Baseline and comparison: For the baseline scheduling policy (`Base`), we use an affinity-based scheduling that maximizes the efficiency of GPU and DLA. Each ML task has a different performance speedup of running on a GPU over a DLA. Given a set of workloads, the scheduler assigns requests to GPU and DLAs based on the speedups of requests on each device. For example, if two ML tasks have GPU speedups over the DLA of 2 and 1, respectively, the task with the higher speedup is assigned to the GPU, as it can better utilize the GPU's capabilities. This affinity-based scheduling has been extensively studied for asymmetric CPUs, such as big and little cores [16]. For fairness, we introduce an additional baseline (`Base fair`) that aims to allocate an equal amount of device time to each ML job. However, due to the less flexible preemption support compared to CPUs and the limited number of DLA contexts, the baseline fair scheduler may not achieve perfect fairness.

In addition to the baseline, we compare our scheme with a version of the same scheduling framework that does not use the interference prediction model. The second configuration, `w/o itf pred`, uses our sampled simulation scheduling but excludes the interference prediction model. In this setup, the execution latency considered during the simulation is the solo execution latency without considering the interference. The third configuration, `w/ itf pred`, represent our full scheduling framework with interference prediction. The final one, `perfect pred` uses the actual profiled interference overhead. Instead of relying on our prediction scheme, this scheduling approach uses the measured interference when the target ML tasks are running concurrently.

Set	Models	% Requests					Possible Interference
		#1	#2	#3	#4	#5	
1	MNASNet 0.5	78.0	3.5	5.0	28.0	67.0	min: 1.08
	MNASNet 1.3	21.5	60.0	50.0	41.5	8.0	max: 1.30
	SqueezeNet 1.1	0.5	36.5	45.0	30.5	25.0	avg: 1.16
2	AlexNet	18.5	52.0	13.5	45.5	43.0	min: 1.09
	MNASNet 1.0	33.5	29.5	70.0	46.0	50.5	max: 1.75
	ShuffleNet V2 x1.0	48.0	18.5	16.5	8.5	6.5	avg: 1.33
3	GoogLeNet	13.0	58.0	23.0	61.5	58.0	min: 1.12
	MobileNet V2	49.5	34.0	29.0	0.5	24.0	max: 2.38
	ShuffleNet V2 x1.0	37.5	8.0	48.0	38.0	18.0	avg: 1.38
4	GoogLeNet	24.5	67.0	36.5	41.0	25.5	min: 1.16
	Inception v3	25.5	12.0	21.5	49.5	0.5	max: 2.69
	ShuffleNet V2 x1.0	50.0	21.0	42.0	9.5	74.0	avg: 1.56
5	AlexNet	3.5	27.5	28.0	30.0	2.0	min: 1.02 max: 1.91 avg: 1.19
	MNASNet 0.5	13.0	17.0	5.5	25.5	3.5	
	ResNet-18	2.0	10.5	11.0	3.0	15.0	
	ShuffleNet V2 x0.5	54.5	28.5	17.0	29.5	11.5	
	SqueezeNet 1.0	17.5	9.0	13.5	5.0	17.5	
	SqueezeNet 1.1	9.5	7.5	25.0	7.0	50.5	
6	Inception v3	0.5	40.0	31.5	20.5	20.0	min: 1.05 max: 2.14 avg: 1.31
	ResNet-18	6.5	10.5	2.0	1.0	3.0	
	ResNet-50	61.0	11.0	13.5	0.5	29.5	
	ShuffleNet V2 x0.5	4.0	7.5	12.5	23.5	2.5	
	ShuffleNet V2 x1.0	13.5	6.0	11.5	50.5	21.5	
	SqueezeNet 1.0	14.5	25.0	29.0	4.0	23.5	
7	GoogLeNet	7.0	38.5	2.5	11.5	7.0	min: 1.07 max: 2.69 avg: 1.34
	Inception v3	12.0	11.5	15.5	7.0	19.0	
	ResNet-18	16.0	10.0	8.5	55.0	20.5	
	ResNet-50	51.5	7.5	20.5	9.0	27.0	
	ShuffleNet V2 x1.0	8.5	16.0	26.5	7.0	11.5	
	SqueezeNet 1.1	5.0	16.5	26.5	10.5	15.0	
8	AlexNet	5.5	3.0	0.5	11.0	27.0	min: 1.09 max: 2.69 avg: 1.46
	GoogLeNet	38.5	6.5	32.0	13.0	16.5	
	Inception v3	32.0	55.0	41.5	6.5	30.5	
	MNASNet 1.3	8.0	12.5	10.0	28.0	14.5	
	ShuffleNet V2 x0.5	1.5	10.0	0.5	19.5	3.0	
	ShuffleNet V2 x1.0	14.5	13.0	15.5	22.0	8.5	

TABLE IV: 40 scenarios: 8 sets with 5 different request ratios

B. *Goodput*

The first scheduling objective is to maximize `goodput` with a given SLO for the execution latency of each ML task. When the SLO for each ML task is specified, `goodput` refer to the throughput of requests that meet the latency constraints set by SLO. Requests that do not meet the SLO are excluded from the goodput counting. This concept of goodput has been used as a performance metric for SLO-constraint ML inference tasks [17].

For each model, we define the SLO in its request latency, which is set to multiples of the solo run latency on a GPU. As shown in the last column of Table III, DLA latency is much slower to execute the ML tasks over GPU in the evaluated platform.

Figure 7 presents the goodputs satisfying the SLO for each ML model. Each bar represent the averaged normalized goodput for a set with 5 request ratio scenarios as shown in Table IV. The SLO in the figure is set to 12x of GPU latency. The figure shows that our scheduler improves upon the baseline by 40% on average. The performance improvements are high in `mix 3`, `mix 4`, and `mix 6` with increases of 58.8%, 93.3%, and 91.4%, respectively.

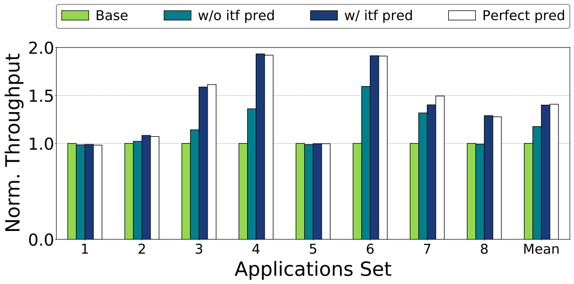


Fig. 7: Goodput normalized to `Base`. SLO is set to 12x of GPU latency

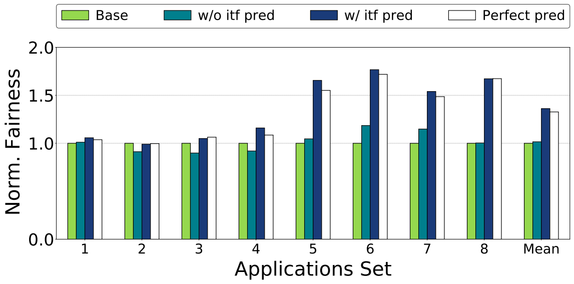


Fig. 8: Maximum throughput under 99% SLO guarantee normalized to `Base`. SLO is 12x latency of GPU run

The figure also highlights the significance of interference prediction. Without the prediction, the performance improvement from `Base` is only 17.6%. Interference prediction significantly enhance the performance. Compared to `w/o itf pred`, the interference prediction offer an 18.1% improvement in goodput. When Compared to the ideal `perfect pred` configuration, `w/ itf pred` provides similar throughput, demonstrating high latency of our interference model. For `mix 2` and `mix 4`, `perfect pred` shows slightly lower performance than `w/ itf pred`, since static perfect prediction may not always be accurate during runtime, when mixes are running dynamically on the platform.

C. *99% SLO Throughput*

The second goal, `99% SLO throughput`, aim to maximize throughput while ensuring that 99% of all requests meet the SLO latency. During the experiments, request rates are increased until 99% of the SLOs can no longer be supported due to high injection rates. The result represents the maximum throughput while still maintaining the 99% SLO compliance. For these experiments, the SLO is set to 12x of the GPU latency.

Figure 8 presents the results for 99% SLO throughput. The figure shows that our scheduling improves the baseline by 36.1% on average. `Mix 5` and `mix 6` exhibit significant improvements of 65.6% and 76.7%, respectively. Compared to `w/o itf pred`, our prediction method achieves substantial enhancements, highlighting the importance of interference prediction. On average, interference prediction improves the non-prediction scheme (`w/o itf pref`) by 33%. For Set 8, the improvement is notably high at 66.7%. Accurate prediction of interference is critical for meeting the SLO requirements.

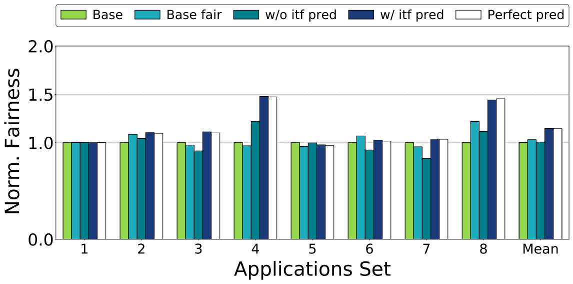


Fig. 9: Fairness normalized to `Base`

Incorrect latency estimations often lead to unpredictable increases in latency, which can significantly impact performance, especially for the 99% SLO goal, where performance is highly sensitive to latency variations.

D. Fairness

The final goal, `fairness`, aims to maximizes the fairness of each DNN model compared to the throughput achieved in an ideal fair run [5]. An ideal fair run represents the performance of the model when it receives equal shares of all device types. The fairness goal is expressed as follows:

$$\text{Maximize}_X \min_m \frac{\text{throughput}(m, X)}{\text{throughput}\left(m, X_m^{\text{equal}}\right)}$$

X_m^{equal} is the allocation given to job *m* assuming it receives an equal share of time on each device in the system. The goal is to find a scheduling that maximizes the minimum normalized performance among jobs.

Figure 9 presents the result with the fairness goal. The figure includes an additional baseline (`Base fair`). `w/o itf pred` often shows worse fairness compared to `Base fair`, as shown in Set 3, 6, and 7, due to the effect of interference. However, the interference-aware scheduling (`w/ itf pred`) can provide better fairness compared to both `Base fair` and `w/o itf pred`. Our scheduling framework improves fairness over `Base fair` due to limitations in the current resource allocation mechanism. `Base fair` attempts to assign equal shares of GPU and DLA time to each task. However, due to coarse-grained context switching support, `Base fair` often fails to provide equal shares of each device to the tasks. In contrast, the simulation-based approach by our scheduler leads to a better fairness support, as the limitation is included in its allocation decision.

VII. Conclusion

This study showed that the interference among heterogeneous processors causes significant performance degradation. It provided a highly accurate prediction model trained from randomly generated machine learning workloads. Using the interference prediction model, the study proposed a goal-independent scheduling framework based on a sampled simulation method.

VIII. Acknowledgments

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) (IITP2017-0-00466 SW StarLab, RS-2024-00339010), funded by the Ministry of Science and ICT (MSIT). It was partly supported by IITP under the Graduate School of Artificial Intelligence Semiconductor grant (IITP-2024-RS-2023-00256472) funded by MSIT

References

[1] W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park, "SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–26, 2021.

[2] M. Han and W. Baek, "HERTI: A Reinforcement Learning-Augmented System for Efficient Real-Time Inference on Heterogeneous Embedded Systems," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021.

[3] P. Radojković, V. Čakarević, M. Moretó, J. Verdú, A. Pajuelo, F. J. Cazorla, M. Nemirovsky, and M. Valero, "Optimal Task Assignment in Multithreaded Processors: A Statistical Approach," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 235–248, 2012.

[4] M. Ditty, A. Karandikar, and D. Reed, "NVIDIA's Xavier SoC," in *Hot chips: a symposium on high performance chips*, 2018.

[5] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[6] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," in *International Symposium on Microarchitecture (MICRO)*, 2011.

[7] F. Romero and C. Delimitrou, "Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018.

[8] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing," in *USENIX Annual Technical Conference (ATC)*, 2022.

[9] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[10] J. Han, S. Jeon, Y.-r. Choi, and J. Huh, "Interference Management for Distributed Parallel Applications in Consolidated Clusters," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 443–456, 2016.

[11] M. Hao, L. Toksoz, N. Li, E. E. Halim, H. Hoffmann, and H. S. Gunawi, "LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[12] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, *et al.*, "DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[13] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[14] H. Vanholder, "Efficient Inference with TensorRT," in *GPU Technology Conference*, vol. 1, pp. 1–24, 2016.

[15] S. Marcel and Y. Rodriguez, "Torchvision the Machine-Vision Package of Torch," in *ACM international conference on Multimedia (MM)*, 2010.

[16] Y. Kwon, C. Kim, S. Maeng, and J. Huh, "Virtualizing Performance Asymmetric Multi-core Systems," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[17] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like Clockwork: Performance Predictability from the Bottom Up," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.