# FairCIM: Fair Interference Mitigation by DNN Switching for Latency-Sensitive Inference Jobs

Seyed Morteza Nabavinejad
*Computer Science Department*
*Worcester Polytechnic Institute*
Worcester, MA
snabavinejad@wpi.edu

Sherief Reda
*School of Engineering*
*Brown University*
Providence, RI
sherief_reda@brown.edu

Tian Guo
*Computer Science Department*
*Worcester Polytechnic Institute*
Worcester, MA
tian@wpi.edu

***Abstract*—It is a common practice to co-locate several jobs, such as DNN inference ones, on a single machine to improve resource utilization, energy efficiency, and overall performance. However, individual job performance might degrade when these jobs compete for shared resources. This degradation can in particular affect latency-sensitive jobs bound by Service Level Agreements (SLA), and result in SLA violations. To tackle this issue and mitigate the impact of interference on latency, we design a system called *FairCIM*. For certain applications such as image classification, various DNNs with varying accuracy and computational complexity are proposed. Upon detection of SLA violations, *FairCIM* dynamically switches between those DNNs to maintain the latency in the presence of co-location interference. To ensure fairness, *FairCIM* selects the jobs for DNN switching based on their resource utilization. We conduct experiments using diverse DNNs, datasets, and hardware platforms to evaluate the *FairCIM* effectiveness. Compared to another approach that uses a resource exchange mechanism, *FairCIM*'s flexibility achieved by DNN switching substantially reduces SLA violations and improves the effective accuracy by up to 244%.**

***Index Terms*—deep neural networks, co-location, interference, fairness**

## I. Introduction

Employment of powerful hardware platforms with enormous computing capacity in data centers and the cloud is on the increase. To gain high resource utilization and energy efficiency in such platforms, a common approach is co-location, where several jobs are co-located on a single machine. While co-location has certain benefits, a varying amount of interference raised among co-located jobs can affect their individual performance in the form of prolonged latency. This increased latency has a particularly detrimental impact on latency-sensitive jobs that need to meet a predefined SLA stated in the form of a latency constraint. Moreover, the interference among co-located jobs has a dynamic nature and can change over time. Therefore, an offline approach that determines the co-location of jobs periodically might not be sufficient to mitigate the impact of interference on the latency of co-located jobs. To address this problem, an approach is needed that can dynamically monitor the latency of jobs and make necessary decisions on the fly to satisfy the SLA.

A large body of prior works has tried to address the problem of co-location interference in different hardware platforms such as CPU-based servers [1], [2] and GPU accelerators [3]–[6]. They are based on exchanging resources between jobs. They assume there are jobs with extra resources and it is possible to take some resources from them and give them to others that experience lower-than-expected performance. While these approaches can achieve promising results, they show limited improvement when all the co-located jobs suffer from low performance, and there is no room for resource exchange.

To alleviate the impact of interference on such cases, e.g., when we have latency-sensitive jobs co-located and there is no room for resource exchange, we need a different approach other than leveraging the resource abundancy. Given that jobs with higher resource utilization have a higher impact on interference, such an approach should consider the resource usage of each job to ensure fairness. To this end, we design and implement *Fair Co-location Interference Mitigation (FairCIM)* that leverages a new control knob to mitigate the interference between co-located latency-sensitive jobs. This control knob is switching between DNNs that belong to the same application (e.g., image classification), but with various accuracy and computational complexities.

*FairCIM* selects a candidate job and switches its DNN with a less complex one to free resources for other jobs. To achieve fairness when selecting the candidate job, *FairCIM* opts for the one that has the highest resource consumption share, instead of simply selecting the one that suffers from SLA violation. To detect the job with the highest share, *FairCIM* considers the features of the input workload of the job, in addition to the computational complexity of its DNN. By switching the DNN of the candidate job to a less accurate one, *FairCIM* trades-off accuracy with latency to reduce the latency of job(s) that is violating its SLA.

We make the following main contributions in this paper:

- Through preliminary experiments, we study the impact of co-location on the latency of DNN inference jobs. We identify two important factors that impact the resource utilization of the jobs, and consequently, interference among them: input workload and DNN computational complexity.
- We introduce the Resource Utilization Score (RUS) metric that quantifies the share of each co-located job from total resource utilization. RUS is calculated based on the

computational complexity of the job's DNN, as well as the distribution of the size of batches and their inter-arrival time in the input workload of the job.

- We design *FairCIM* approach to mitigate the impact of interference on the latency of co-located jobs. *FairCIM* uses DNN downgrading to maintain the latency, which means replacing the DNN of the jobs with a less complex one to free up some resources. In the case of SLA violation, *FairCIM* selects the job with the highest RUS for DNN downgrading. Since DNN downgrading can affect accuracy, *FairCIM* selects the job with the highest RUS to achieve fairness. We will open-source the relevant research artifacts upon acceptance.

We implement *FairCIM* and deploy it on a CPU-based server and a GPU accelerator to show its generality. We conduct experiments using well-known DNNs and datasets for image classification, video saliency, and natural language processing to study the efficacy of *FairCIM*. We compare the performance of *FairCIM* with two state-of-the-art approaches, CLITE [2] for CPU-based server and C-Laius [4] for GPU accelerator. The results show that compared with CLITE, *FairCIM* can yield better performance. It can successfully maintain the latency of jobs, and achieve up to 244% (98% on average) improvement in effective accuracy.

The organization of the rest of the paper is as follows: In Section II we provide background on the impact of co-location on the latency of DNN inference and also study the impact of input workload on the resource utilization of jobs. The problem statement and formulation and our proposed approach are described in Section III and Section IV respectively. We present evaluation results in Section V and discuss the related work in Section VI. Finally, we conclude the paper in Section VII.

## II. Background and Motivation

### A. Co-location Contention

The Computing power and memory capacity of hardware platforms used in data centers and the cloud is increasing. Popular workloads, including DNN inference jobs, deployed on these platforms might not fully utilize the available resources all the time individually. Therefore, it is a common practice to co-locate several jobs on a single machine to improve resource utilization. Via preliminary experiments, we study the impact of co-location on the performance of DNN inference jobs by quantifying the latency increase resulting from interference among them. We use six DNNs (Mobilenet-V1-1, Inception-V1, Inception-V4, NASNet-Large, ResNetV2-101, ResNetV2-152). First, we launch them individually (without co-location) to measure their average latency (for processing one input at a time). Then, we co-locate them all together and measure the average latency again. We co-locate all six DNNs to maximize resource utilization, resulting in resource saturation and inevitable interference among them.

The results are shown in Fig 1. We observe that co-location results in prolonged latency for all of them; however, the amount of latency increase varies from DNN to DNN. The more complex and larger ones such as ResNet-V2-152 and Inception-V4 experience higher latency increases than the smaller and less complex ones such as Inception-V1 and MobileNet.

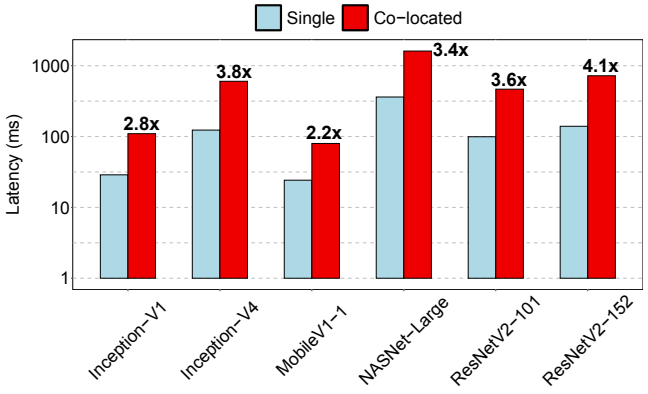



Fig. 1: Impact of co-location interference on latency of various DNNs. The numbers on top of the bars show the latency increase of co-location compared with a single deployment. The Y-axis is shown in the base-10 log scale.

### B. DNN Input Workload

The input workload of DNN inference jobs typically comprises a continuous stream of request batches with varying sizes and inter-arrival times [7], [8]. In addition to the computational complexity of the jobs' DNNs, these fluctuating workloads also impact the resource utilization pattern of the jobs. To illustrate this point, we examine two DNNs with significantly different computational complexities: Inception-V1 with 13.22 MFLOPs and Inception-V4 with 91.94 MFLOPs computational complexity. These DNNs are deployed on the server individually (without co-location) and are fed with two distinct input workloads. The batch size and inter-arrival time distribution for each DNN's input workload are shown in Fig. 2(a).

The input workload of Inception-V1 consists of bigger batches arriving at higher rates (shorter inter-arrival time) compared with Inception-V4. The average batch size for Inception-V1 is 64 and for Inception-V4 is 4. The average inter-arrival time for Inception-V1 is 150 ms and for Inception-V4 is 750 ms. Analyzing the CPU utilization of each DNN's input workload reveals that while Inception-V4 is bigger than Inception-V1 (around 7x more complex), its resource utilization is lower due to its input workload. Therefore, we conclude that the resource utilization of DNN inference jobs depends on both the input workload and DNN's computational complexity.

## III. Problem Statement

### A. Effective Accuracy

Before presenting the problem formulation, we introduce the performance metric we use in this work to evaluate the efficacy

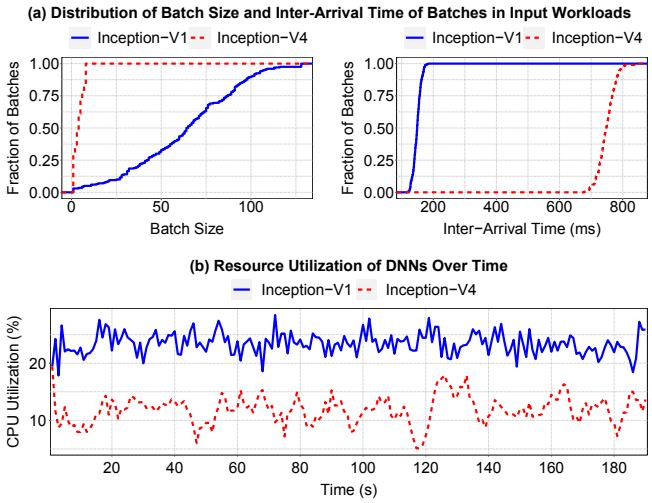



Fig. 2: Impact of input workload on resource utilization of DNNs. While Inception-V1 is smaller and less computationally complex than Inception-V4, it utilizes CPU more since its input workload consists of bigger batches that arrive at a higher rate.

of different approaches. Zhang *et. al.* [8] argue that for ML-as-a-Service (MLaaS) latency or accuracy alone cannot be used to measure the satisfaction of users, but both of them should be considered simultaneously. To his end, they introduce the *Effective Accuracy* metric. Denoting the accuracy of results by *acc* with a range from 0 (zero percent accuracy) to 1 (100% accuracy) and the fraction of requests that meet their SLA (latency constraint) by $req_{SLA}$, the effective accuracy (EA) is obtained as follows:

$$EA = req_{SLA} \times acc \tag{1}$$

Our proposed approach can meet the latency constraint by sacrificing accuracy. However, other approaches that we compare *FairCIM* against, do not affect accuracy but violate the SLA more frequently. Therefore, for the sake of fair comparison, we employ effective accuracy metric as it considers both SLA violation and accuracy reduction.

### B. Problem Formulation

There are several DNN inference jobs co-located on a single machine. Each job has its latency SLA which is specified by the user during job submission. For one or more job(s), more than one DNN is available for application. The available DNNs do the same task but with different computational complexity (i.e., resource requirement) and accuracy. We can change the DNN of those jobs during runtime to manage the latency, but we should be aware of their impact on accuracy. Therefore, we use effective accuracy metric to consider both latency and accuracy. The objective function of the problem is maximizing the average effective accuracy of all the co-located jobs, subject to maintaining the SLA of each job:

$$\begin{aligned} &Maximize \quad \frac{\sum_{i \in N} EA_i}{N} \\ S.T. \quad &\forall i \in N, \quad Latency_i < SLA_i \end{aligned} \tag{2}$$

The number of co-located DNNs is denoted by N. The SLA of jobs is defined in the form of tail latency. It means the jobs can tolerate a certain amount of latency violation, as long as their tail latency is less than or equal to the SLA.

## IV. Methodology

In the following, we present our proposed approach to address the problem stated and formulated in Section III. First, we introduce Resource Utilization Score (RUS). After that, we describe the architecture of *FairCIM*. The overall flow of *FairCIM* is shown in Fig. 3 and its pseudo-code is presented in Algorithm 1.

### A. Resource Utilization Score

We observed in Section II-B that three parameters determine the resource utilization intensity of a DNN inference job: the average batch size of input workload (ABS), the average time between the arrival of the batches (AAT), and the computational complexity of the DNN (DNNCC). Therefore, considering these three parameters, we can compare the resource utilization of different jobs co-located with each other. To make the comparison comprehensive, we need a metric that can *estimate* the resource utilization of jobs based on the parameters mentioned above. The metric should be easy to obtain and not impose significant overhead on the system. To this end, we introduce Resource Utilization Score (RUS). RUS can be calculated as follows:

$$RUS = \frac{1}{AAT} \times ABS \times DNNCC \tag{3}$$

In (3), the $\frac{1}{AAT}$ shows the number of batches received by a job in time unit (assuming AAT is in seconds. $\frac{1000}{AAT}$ if AAT is in milliseconds). The lower the AAT, the higher the job's load in the time unit (assuming the same ABS). RUS allows us to consider the three aforementioned parameters *simultaneously* to compare the resource utilization of different jobs and identify the ones with more resource utilization than others. While our observations show that RUS can successfully estimate the resource utilization of jobs, it is possible to use other metrics with different definitions for resource utilization estimation and implement them in FairCIM. For example, it is possible to consider different priorities for jobs and consider it in the definition of RUS. In the current definition, we assume the same priority for all the jobs.

### B. FairCIM Design and Implementation

As mentioned earlier in Section I, the resource exchange mechanism of previous approaches is not completely effective when all the co-located jobs are latency-sensitive and there are no extra resources to take from one job and give to another. Therefore, we should employ a new control knob to manage the resource utilization and maintain the latency. Of the three

parameters that affect resource utilization (AAT, ABS, and DNNCC), the first two depend on input workload and we do not have control over them. Therefore, the only remaining option is controlling the DNN computational complexity.

For some popular applications such as image classification, various DNNs are developed (e.g., Inception, ResNet, MobileNet) that do the same thing, but with different accuracy and computation cost. If one (or more) of the jobs in the set of co-located ones is executing an application with such a feature (several DNNs), we can control its resource utilization by changing its DNN. This is the control knob that we leverage in the design of *FairCIM*. In the following, we discuss the *FairCIM* design in detail.

*FairCIM* constantly checks the jobs' latency. Upon detecting a job with latency higher than SLA, the interference mitigation mechanism is activated. At first, based on the history of the input workload of each job, its AAT and ABS are calculated. Different frameworks such as TensorFlow have their interface For calculating the computational complexity of DNNs. Having all the necessary parameters, *FairCIM* calculates the RUS of each job according to (3). The job with the highest RUS is a potential candidate for DNN downgrading (replacing its DNN with a less complex one). *FairCIM* checks to see if DNN downgrading is possible for this job. For some jobs, there might be no alternative DNN to replace with their current one. For some others, it is possible that the current DNN being used is the least complex one. In these cases, *FairCIM* skips the current candidate job and moves to the next one with the highest RUS. It continues the search until there is one job with the DNN downgrading possibility, or there is no more job to check.

When finding the candidate job, it is time for DNN replacement. Downgrading the DNN can reduce the accuracy of the job as the less complex DNNs usually have less accuracy as well. Therefore, when several less complex DNNs are available for replacement, *FairCIM* opts for the one with the closest complexity to the DNN currently used. For instance, when DNN1 and DNN2 with computational complexity (MFLOPs) of 70 and 90 are available to be replaced by the current DNN, *FairCIM* selects the DNN2 that has higher computational complexity. In this way, *FairCIM* aims to minimize the accuracy loss of the job. After changing the DNN of the candidate job, *FairCIM* continues monitoring the latency of the jobs to detect possible SLA violations, and if detected, repeats the steps. Moreover, if FairCIM detects that the resource utilization is very low and there is room for higher accuracy, it can upgrade the DNN of job(s) to improve accuracy. We implemented *FairCIM* using Python on top of TensorFlow. While we use TensorFlow, *FairCIM* is compatible with other frameworks too. *FairCIM* can also be deployed on any hardware platform that supports co-location including CPU-based and GPU-based servers.

A possible alternative to *FairCIM* is directly downgrading the DNN of the job(s) facing SLA violations, ignoring resource utilization of jobs. However, this approach lacks fairness, as smaller jobs with lower resource utilization could suffer accuracy loss while larger jobs with high resource utilization can evade consequences. *FairCIM*, prioritizing fairness, selects jobs for DNN downgrading based on RUS, ensuring that jobs with higher resource utilization experience more accuracy loss in the presence of SLA violations. *In other words, FairCIM ensures fairness as follows: The jobs with higher resource utilization would experience more accuracy loss in the presence of SLA violations, even if they do not face SLA violation themselves and it is the other jobs that suffer from it. It is the price they should pay for consuming more resources than others.*

**Algorithm 1** *FairCIM* Pseudo-code

**Input:** set of DNN jobs (JS) and their SLAs

1: **while** True **do**
2: **for** $i \in JS$ **do**
3: **if** $Latency_i > SLA_i$ **then**
4: SLAViolation = True
5: **if** *SLAViolation == False* // no job is violating its SLA **then**
6: Continue from line1
// Proceeding to this line means SLA violation has happened
7: **for** $i \in JS$ **do**
8: $AAT_i$, $ABS_i$ = profile (input workload)
9: $DNNCC_i$ = profile ($DNN_i$)
10: $RUS_i = \frac{1}{AAT_i} \times ABS_i \times DNNCC_i$
11: JS.Sort(in descending order with respect to "RUS")
// Now the JS(1) contains the job with highest RUS
12: **for** $i \in JS$ **do**
13: **if** $DNN_i$ is the smallest possible one **then**
14: continue //there is no room for downgrading the DNN of job
15: **else**
16: CJ = JS(i) // Candidate Job
17: break // break the for loop and go to next step
18: **if** Candidate == NULL **then**
19: There is no room for downgrading the DNN of any job
20: go to line 1
21: $DNN_{CJ} = Max_{DNNCC}$ ($\forall i \in$ DNNSet, $DNNCC_i < DNNCC_{CJ}$)

## V. Evaluation

In this section, we evaluate the effectiveness of *FairCIM* regarding the effective accuracy improvement of co-located DNN inference jobs. Specifically, we are interested in understanding how much the new control knob of changing the DNN model can improve the effective accuracy compared with the traditional approach of exchanging resources between jobs. To this end, we implement FairCIM and two prior approaches, CLITE [2] and C-Laius [4]; and deploy them on two different hardware platforms and use several DNN models and datasets for evaluation.

### A. Experimental Setup

**Hardware Platforms.** We run experiments on two platforms, CPU-based and GPU-based servers, to show the applicability of our approach on different hardware: 1) A dual-socket server equipped with two Xeon processors, each containing 28 cores operating at 2.4 GHz, and 128 GB of DDR4 memory. 2) A Tesla P40 GPU accelerator featuring 3840 CUDA cores with a peak frequency of 1531 MHz and a total memory capacity of 24 GB GDDR5.

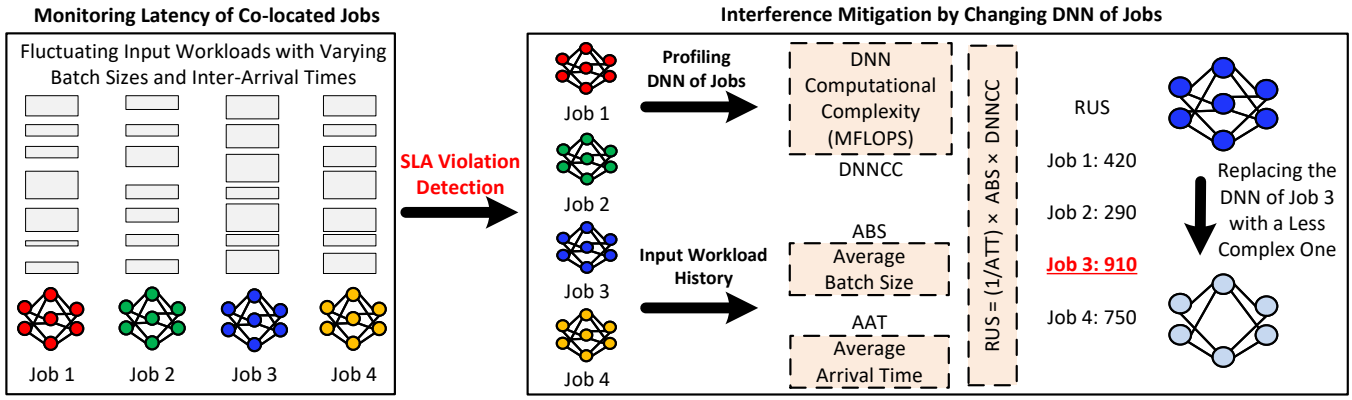



Fig. 3: Overall flow of FairCIM.

**Models and Datasets**. We use DNNs from different domains, from computer vision to natural language processing to video saliency, with various computing requirements and datasets. The networks and datasets are listed in Table I. Computer vision, and in particular image classification, is a popular field of study and numerous DNNs are designed for it with varying computational complexities and accuracy. Therefore, we can use them to show how *FairCIM* mechanism of DNN downgrading works in practice. We employ 16 image classification networks with two datasets in the experiments: ImageNet [9] and CalTech-256 [10]. The computational complexity (DNNCC) of image classification DNNs is also listed in Table I. Note that for TextClassif and DeepVS DNNs, we could not obtain computational complexity as the tool we used does not support them.

From the natural language processing (NLP) domain, we employ a DNN for text classification [11], which we call TextClassif. For the input data of this DNN, we use Sentiment140 [12] and IMDB Reviews [13] datasets. DeepVS [14] is another DNN we use in our experiments that targets video saliency prediction. For these two DNNs (TextClassif and DeepVS), we do not have alternative DNNs. Therefore, for the jobs that use them, the DNN downgrading is not possible. We use these DNNs in our experiments to show how *FairCIM* acts as such DNNs are present among co-located jobs.

**Workloads**. We use eight co-location sets to evaluate the efficacy of *FairCIM* and compare its performance against other approaches. The sets are synthesized to cover various scenarios, from low interference to high interference among co-located jobs, and from jobs with many alternative DNN models to jobs with few or no alternative DNN models. Five co-located jobs are considered in each set to make sure that the resource utilization reaches its maximum capacity and interference among co-located jobs happens. The SLA of each job is selected such that 1) it would be tight enough that some of the jobs in a set cannot meet it and face SLA violation and 2) it would be relaxed enough that a modification approach can help mitigate the impact of co-location and address the SLA violation. The co-location sets are listed in Table II.

TABLE I: List of DNNs Used in the Experiments

| DNN (Abbreviation) | Domain | Dataset (Size) | DNNCC (MFLOPs) |
|---|---|---|---|
| Inception-V1 (Inc-V1) [15] | Image Classification | ImgaeNet & Caltech-256 | 13.22 |
| Inception-V2 (Inc-V2) [16] | | | 22.34 |
| Inception-V3 (Inc-V3 [17] | | | 54.25 |
| Inception-V4 (Inc-V4) [18] | | | 91.95 |
| MobilenetV1-1 (MobV1-1) [19] | | | 8.42 |
| MobilenetV1-05 (MobV1-05) [19] | | | 2.64 |
| MobilenetV1-025 (MobV1-025) [19] | | | 0.93 |
| MobilenetV2-1 (MobV2-1) [20] | | | 6.94 |
| MobilenetV2-14 (MobV2-14) [20] | | | 12.12 |
| NASNET-A-Large (NAS-Large) [21] | | | 177.12 |
| NASNET-Mobile (NAS-Mob) [21] | | | 10.51 |
| PNASNET-Large (PNAS-Large) [22] | | | 171.77 |
| PNASNET-Mobile (PNAS-Mob) [22] | | | 10.06 |
| ResNet-V2-50 (ResV2-50) [23] | | | 51.01 |
| ResNet-V2-101 (ResV2-101) [23] | | | 88.89 |
| ResNet-V2-152 (ResV2-152) [23] | | | 120.08 |
| TextClassif (-) [11] | NLP | Sentiment140 [12] IMDB [13] | - |
| DeepVS (-) [14] | Video Saliency | LEDOV [14] DHF1K [24], [25] | - |

**System Comparison.** We compare the performance of *FairCIM* against CLITE [2] for the CPU server. CLITE uses the resource exchange mechanism to maintain the latency by taking the resources from the one job(s) and allocating them to other(s) that suffer from SLA violation. It takes the resources from the job with the highest number of CPU cores and allocates them to other jobs that experience SLA violations. For the GPU accelerator, we compare *FairCIM* with C-Laius [4] that has a similar mechanism to CLITE but for GPUs.

### B. Experimental Results

*1) FairCIM vs. CLITE: Effective Accuracy Analysis:* The effective accuracy results of FairCIM and CLITE across various sets and their respective jobs are illustrated in Fig. 4. Notably, *FairCIM* generally attains higher effective accuracy than CLITE in most sets and jobs. On average, across all sets,

TABLE II: Specification of Co-location Sets Used in the Experiments

| Set 1 | | | | Set 2 | | | | Set 3 | | | | Set 4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) |
| 1 | Inc-V1 | ImageNet | 320 | 1 | Inc-V2 | ImageNet | 240 | 1 | MobV1-1 | ImageNet | 238 | 1 | Inc-V1 | CalTech | 140 |
| 2 | Inc-V4 | ImageNet | 1550 | 2 | Inc-V3 | ImageNet | 510 | 2 | MobV1-025 | ImageNet | 160 | 2 | NAS-Mob | ImageNet | 290 |
| 3 | MobV2-14 | CalTech | 420 | 3 | MobV1-05 | ImageNet | 150 | 3 | MobV2-1 | CalTech | 218 | 3 | PNAS-Mob | CalTech | 260 |
| 4 | PNAS-Large | CalTech | 1850 | 4 | MobV2-14 | ImageNet | 310 | 4 | NAS-Large | CalTech | 2746 | 4 | ResV2-152 | CalTech | 1071 |
| 5 | ResV2-101 | CalTech | 710 | 5 | PNAS-Large | CalTech | 1600 | 5 | ResV2-50 | CalTech | 285 | 5 | MobV2-1 | ImageNet | 224 |
| **Set 5** | | | | **Set 6** | | | | **Set 7** | | | | **Set 8** | | | |
| Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) | Job # | DNN | Dataset | SLA (ms) |
| 1 | Inc-V4 | CalTech | 620 | 1 | Inc-V2 | CalTech | 220 | 1 | Inc-V3 | CalTech | 400 | 1 | ResV2-152 | ImageNet | 1542 |
| 2 | MobV1-1 | CalTech | 206 | 2 | MobV1-025 | CalTech | 105 | 2 | MobV1-05 | CalTech | 126 | 2 | Inc-V4 | ImageNet | 1654 |
| 3 | NAS-Mob | CalTech | 309 | 3 | NAS-Large | ImageNet | 2962 | 3 | PNAS-Mob | ImageNet | 400 | 3 | Inc-V3 | CalTech | 800 |
| 4 | ResV2-50 | ImageNet | 568 | 4 | PNAS-Large | ImageNet | 3690 | 4 | ResV2-152 | ImageNet | 1399 | 4 | TextClassif | IMDB | 82 |
| 5 | ResV2-101 | ImageNet | 530 | 5 | TextClassif | Sentiment | 55 | 5 | DeepVS | DHF1K | 1750 | 5 | DeepVS | LEDOV | 2064 |

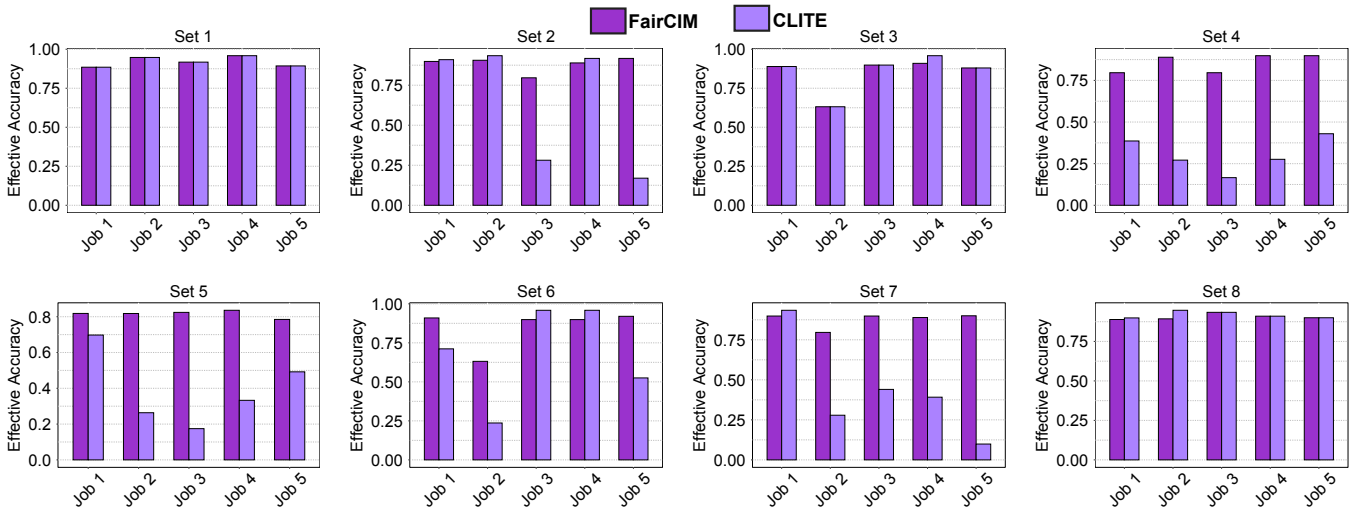



Fig. 4: Effective Accuracy of Jobs in each Set

*FairCIM* improves the effective accuracy by 98% compared with CLITE, with the highest improvement occurring in Set 7 with 244%. When *FairCIM* changes the DNN of a job, the accuracy reduction is moderate. For example, based on our measurements, Inception-V4 achieves 94.7% accuracy on the ImageNet dataset, while Inception-V3 achieves 93.4% accuracy, resulting in only a 1.3% accuracy loss. Downgrading the DNN significantly enhances the number of requests meeting the SLA, $req_{SLA}$. The significant improvement of $req_{SLA}$ compensates for moderate accuracy loss caused by DNN downgrading, leading to high effective accuracy in *FairCIM*.

Unlike *FairCIM*, CLITE tries to address the SLA violation by its resource exchange mechanism. This approach cannot be effective all the time, especially when the other jobs can hardly meet their SLA and have no extra resources. Consequently, some jobs can only process a small number of requests within their latency constraint, resulting in very low $req_{SLA}$. Although CLITE skips DNN downgrading and the associated accuracy loss faced by FairCIM, the low $req_{SLA}$ diminishes the final effective accuracy.

CLITE achieves slightly higher effective accuracy than *FairCIM* in Sets 3 and 8 by less than 2%. In these Sets, some jobs have surplus resources beyond their need to meet SLA. It allows CLITE to redistribute these extra resources to other jobs facing SLA violations. This resource exchange procedure helps CLITE maintain the latency of all jobs. *FairCIM*, however, needs to change the DNNs for specific jobs (e.g., Job 1 and Job 2 in Set 8) to uphold the latency for all the jobs, resulting in lower accuracy, and consequently, lower effective accuracy.

In Sets 2 and 6, CLITE can achieve higher effective accuracy in a couple of jobs, but its overall effective accuracy is lower. In these sets, the resources are only sufficient for a few jobs to meet their SLA, with no additional resources for others. *FairCIM* downgrades the DNN of most jobs in these sets, leading to lower accuracy in some jobs compared to CLITE with the same $req_{SLA}$. In these jobs, the effective accuracy of *FairCIM* is lower than CLITE. In Sets 6 and 7, *FairCIM* can manage the latency of TextClassif and DeepVS jobs (Job 5 in both sets) while they do not have alternative DNN. It shows the effectiveness of *FairCIM* in the presence of such jobs. Finally, in Set 1, no job violates its SLA and both approaches achieve the same effective accuracy.

*2) FairCIM's Dyanmic DNN Adaptation:* In this section, we study the number of DNN changes in each set performed by *FairCIM* to understand how it leverages DNN downgrading

to maintain latency. The detailed information is shown in Table III. The numbers show how many times the DNN of a job is downgraded over time. The number of changes in Sets 2, 4, and 7 is very high. The reason is more significant SLA violations in these sets: either one job with significant SLA violations or several jobs with SLA violations. Therefore, *FairCIM* changes the DNN of the jobs more frequently to address the SLA violations. Unlike *FairCIM*, CLITE cannot address the SLA violations in those sets as all of the jobs suffer from it, and hence, CLITE achieves low effective accuracy.

Another observation is that some jobs experience more DNN downgrading than others. For instance, the DNN of Job 4 has been changed 10 times in Set 7, while Job 3 has only experienced it twice, and the DNN of Jobs 2 and 5 has not changed at all. The reason for this excessive DNN downgrading of some jobs is their high RUS. In Fig. 5, we show the flow of DNN change for Set 5 to see how the DNNs are changed over time, based on RUS of jobs. We use Set 5 as the number of changes is moderate and can be clearly shown in a figure. In the first three steps, Job 1 has the highest RUS, and hence, its DNN is changed. At step 4, the RUS of Job 1 becomes less than Job 5. Therefore, at that step Job 5 is selected. At the final step, Job 1 again has the highest RUS, and hence, it is selected for DNN downgrading. After this step, all the jobs can meet their SLA and no further SLA violation occurs. Note that since the input workload of jobs is not changing, their ABS and AAT remain the same over time and their RUS only changes when their DNN is changed (i.e., their DNNCC). Hence, after each DNN change, only RUS of the job with DNN downgrading is changed and the RUS of the other ones remains intact.

*3) FairCIM and CLITE In-Depth Comparisons:* In the following, we discuss the results for latency of jobs for *FairCIM* and CLITE. Due to lack of space, we only show the results for one set (Set 5). The latency of jobs under both approaches and their SLA is shown in Fig. 6 for a period of time, and the distribution of requests' latency is shown in Fig. 7. *FairCIM* effectively addresses the SLA violations occurring in jobs over time by employing its DNN downgrading mechanism.

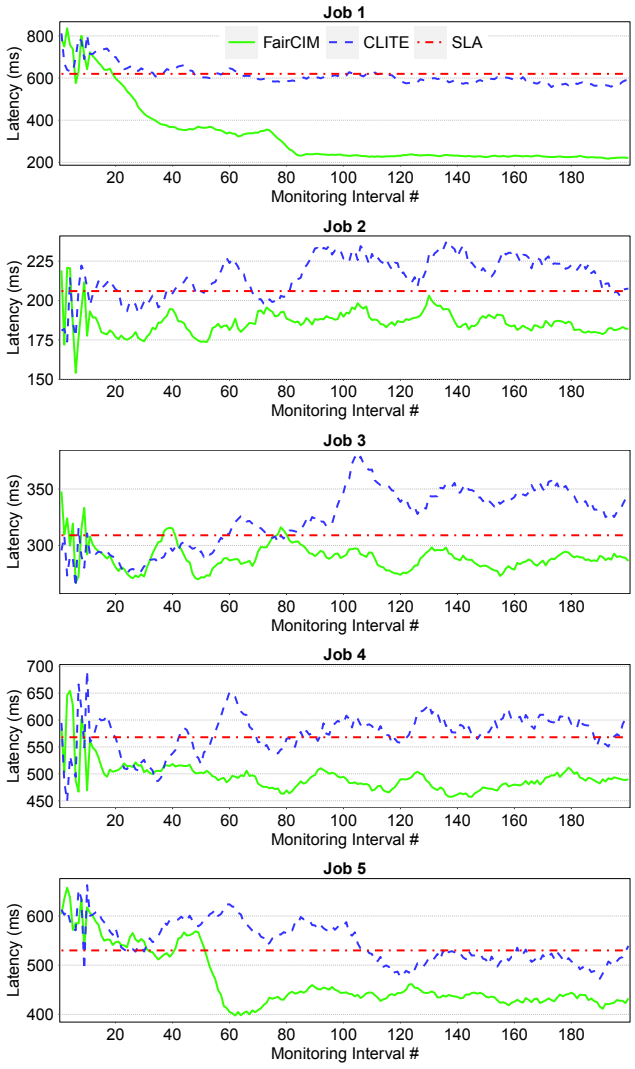



Fig. 6: Impact of different approaches on the latency of jobs and SLA violation for Set 5.

TABLE III: Job's DNN Downgrading Frequency by FairCIM

| | Set 1 | Set 2 | Set 3 | Set 4 | Set 5 | Set 6 | Set 7 | Set 8 |
|---|---|---|---|---|---|---|---|---|
| Job 1 | - | 6 | - | 6 | 4 | - | 8 | 3 |
| Job 2 | - | 5 | - | 2 | - | - | - | 1 |
| Job 3 | - | - | - | 3 | - | 2 | 2 | - |
| Job 4 | - | 3 | 7 | 11 | - | 1 | 10 | - |
| Job 5 | - | 8 | - | - | 1 | - | - | - |
| Total | 0 | 22 | 7 | 22 | 4 | 3 | 20 | 4 |

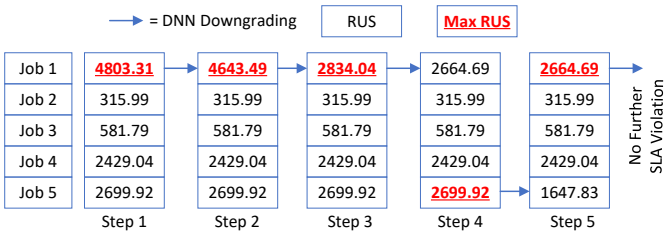
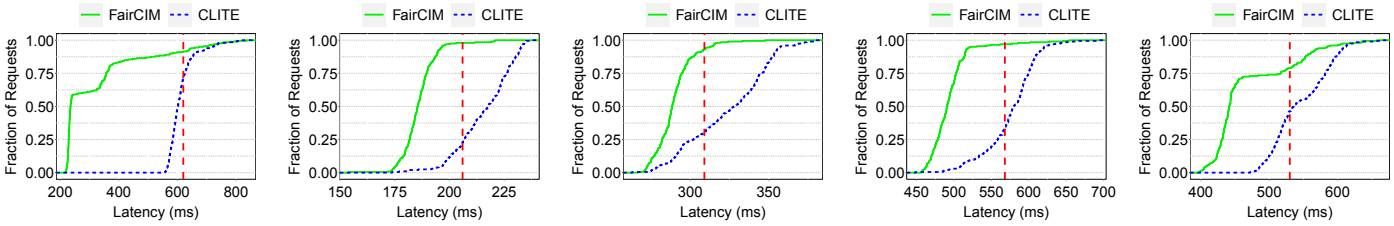



Fig. 5: FairCIM flow of DNN downgrading of jobs over time for Set 5.

Downgrading the DNN of Job 1 (see Fig. 5 for details) not only reduces its latency (as it uses less complex DNN) but also affects the latency of other jobs, e.g., Job 5. While the latency of Job 5 decreases as a result of DNN change in Job 1, it still suffers from SLA violation. Therefore, after changing the DNN of Job 1 several times, *FairCIM* decides to downgrade the DNN of Job 5 around Monitoring Interval 50 as it has the highest RUS at the moment (Fig. 5, Step 4). Over time, Job 3 again experiences SLA violation briefly around Monitoring Interval 80. At this time, *FairCIM* changes the DNN of Job 1 for the last time to address the SLA violation of Job 3. Note that while Job 3 has SLA violation at this step, *FairCIM* decides to downgrade the DNN of Job 1 as this job has the maximum RUS at the time. This is an example of fairness of *FairCIM* that targets the jobs with high resource utilization.




(a) Job 1 (b) Job 2 (c) Job 3 (d) Job 4 (e) Job 5

Fig. 7: Latency distribution of requests for each job in Set 5 under FairCIM and CLITE. The vertical red dashed line indicates the latency constraint (SLA) of each job.

CLITE, on the other hand, cannot address the SLA violation properly and some jobs face significant SLA violations at any time. Since no job with extra resources that can meet its latency constraint with fewer resources is present in the sets, the CLITE exchanging the resources only shifts the problem from one job to another. For example, CLITE takes a portion of resources from Jobs 2, 3, and 4 and gives them to Jobs 1 and 5 to reduce their latency. While it is successful and Jobs 1 and 5 no longer violate their SLA significantly, Jobs 2, 3, and 4 start experiencing high levels of SLA violation.

### C. Sensitivity Analysis

We conduct another set of experiments to study the impact of SLA on the performance of *FairCIM*. Earlier, we saw that for Set 1, *FairCIM* does not change the DNN of any job, as they can meet their SLAs with their initial DNNs. To examine the sensitivity of *FairCIM* to the SLA of jobs, we change the SLA of Job 5 of that set and make it tighter step by step. The results for DNN downgrading of each job are shown in Table IV and the effective accuracy of each job is shown in Fig. 8. The table shows how many times the DNN of each job in the set is changed under different SLAs for Job 5, and the effective accuracy of each job for different SLAs is shown in the bar chart. As the SLA becomes tighter, *FairCIM* changes the DNN of jobs more frequently to maintain the latency. When SLA is 639 ms, the DNNs of jobs are only changed three times, but for the tightest SLA (213 ms), DNN downgrading is performed 16 times by *FairCIM*.

We see in Table IV that while it is Job 5 that is facing a tighter SLA, *FairCIM* does not necessarily start DNN downgrading from it. Instead, it starts from jobs with higher resource utilization (Job 2 and Job 4). It is only after SLA becomes very tight that *FairCIM* starts downgrading the DNN of Job 5 as well. Consequently, the reduction of effective accuracy does not only happen in Job 5, but it is spread across the jobs. Hence, instead of seeing a significant effective accuracy reduction in Job 5, we see a moderate reduction across all the jobs. Since Job 2 and Job 4 are utilizing more resources, their effective accuracy reduction is more significant than Job 1 and Job 3. These results demonstrate the ability of *FairCIM* to adapt itself to the different SLAs by changing the frequency of DNN downgrading across the co-located jobs. It also shows how *FairCIM* can practice fairness by selecting the jobs for DNN downgrading based on their resource utilization.

TABLE IV: Sensitivity Analysis of FairCIM to SLA of Jobs for Set 1

| | SLA of Job 5 (Relaxed to Tight) | | | | |
|---|---|---|---|---|---|
| | 710 ms | 639 ms | 497 ms | 355 ms | 213 ms |
| Job 1 | - | - | - | - | 1 |
| Job 2 | - | 2 | 2 | 4 | 5 |
| Job 3 | - | - | - | - | 1 |
| Job 4 | - | 1 | 1 | 4 | 6 |
| Job 5 | - | - | 1 | 3 | 3 |
| Total | - | 3 | 4 | 11 | 16 |

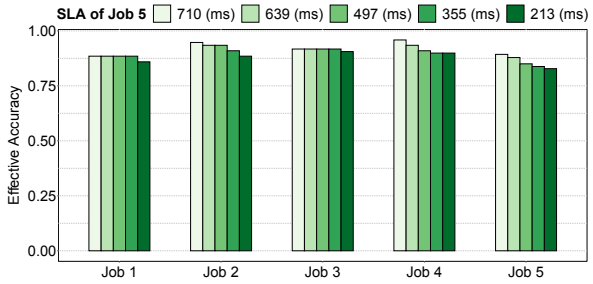



Fig. 8: Impact of changing the SLA of Job 5 on the effective accuracy. As SLA becomes tighter, effective accuracy reduction happens for all the jobs. However, the jobs with higher resource utilization (Job 2 and 4) face more significant decrease in effective accuracy

### D. FairCIM Evaluation on GPU

With the emergence of Deep Neural Networks (DNNs) the employment of GPU accelerators for providing significant computing resources for such networks is on the rise. Since one application might not be able to fully utilize a GPU accelerator, it is a common practice to co-locate several applications on a single GPU as well. Therefore, we extend our experiments to study the efficacy of *FairCIM* for GPU accelerators. To this end, we use a Tesla P40 GPU accelerator that is introduced in Section V-A, Hardware Platforms. We compare the *FairCIM* against the C-Laius approach introduced in Section V-A, System Comparison. The sets are similar to

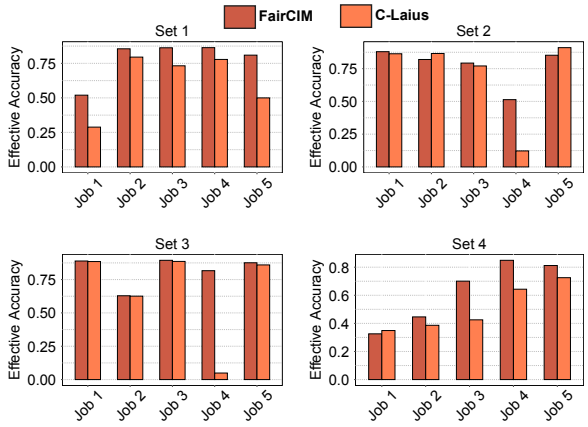



Fig. 9: FairCIM and C-Laius effective accuracy results for GPU accelerator.

the ones used for the CPU server (see Table II). However, the SLAs need to be updated as the requests are processed faster on GPU. Therefore, we reduce the value of SLAs for jobs. Due to lack of space, we show the results for effective accuracy of 4 sets in Fig. 9. *FairCIM* can outperform C-Laius, similar to CLITE, as C-Laius design also depends on the availability of extra resources for exchange between jobs. *FairCIM* improves the effective accuracy by up to 313% (on average 108%). These results indicate the effectiveness of *FairCIM* for GPU accelerators.

### *E. Discussion*

The primary overhead FairCIM introduces to the system comes from DNN switching. When a job's DNN needs to be changed, the system must terminate the job and restart it with a new DNN. Deploying this new DNN on the hardware takes time, which varies based on the DNN's size and the hardware's computing power. In our tests, it takes approximately 30 seconds on average to terminate a job and resume it with the new DNN. During this period, the job cannot process incoming requests, which must wait in the queue.

## VI. Related Work

Enhancing the efficiency of DNN inference jobs deployed on different hardware platforms is a prominent area of research focus that is widely studied in previous works [26]–[30]. Among these works, a large sub-category has studied the advantages and challenges of co-location of DNN jobs [31]–[36]. The study of PERSEUS [37] and Jain *et. al.* [38] showed that while co-location can help to improve the throughput and resource utilization, it negatively affects the latency. Therefore, the focus of a large body of previous works is on co-locating jobs in a manner that their competition of shared resources is minimized, and hence, have no or little interference with each other [27], [39], [40]. These approaches do not offer any method for mitigating the interference among co-located jobs during runtime.

Another category of previous works addresses co-location interference by designing and implementing runtime systems. Baymax [3] and C-Laius [4] try to mitigate the impact of co-location on the latency of latency-sensitive jobs that share the GPU accelerator with throughput-oriented jobs. They aim to maximize the throughput of the latter while meeting the latency of the former by re-allocationing time slots [3] or computing resources [4] of GPUs. In addition to GPUs, the co-location of throughput-oriented and latency-sensitive jobs on CPU servers is studied in previous works. The earlier works consider the co-location of one latency-sensitive job with one or more throughput-oriented job(s) [41]–[43]. The more recent works, however, consider the co-location of several latency-sensitive jobs with one or more throughput-oriented ones [1], [2]. They use hardware/software resource allocation methods to assign the resources to jobs dynamically at runtime to meet the SLA of latency-sensitive jobs while maximizing the throughput of other ones.

The main issue of all the previous works that focus on mitigating co-location interference is their assumption of the availability of one or more throughput-oriented job(s) among the co-located ones. As latency is not crucial for such jobs, it is possible to invoke some of their resources and assign them to latency-sensitive jobs that face SLA violations. This assumption renders their effectiveness low when they are employed for co-location sets consisting of only latency-sensitive jobs. Taking resources from one latency-sensitive job to give them to another, puts the former at the risk of SLA violation. So, the SLA violation would only be shifted from one job to another, instead of being addressed. Our *FairCIM* approach can address this shortcoming of previous works by leveraging DNN downgrading, instead of employing a direct resource exchange mechanism.

## VII. Conclusion

In this paper, we introduced a new approach called *FairCIM* for mitigating the impact of interference on the performance of co-located DNN inference jobs. *FairCIM* addresses the drawback of the previous approaches that are based on resource exchange. Therefore, they fail to maintain SLA when none of the co-located jobs has extra resources. *FairCIM* leverages switching between diverse DNNs developed for the same application to maintain the latency, while slightly affecting the accuracy of the results. Furthermore, *FairCIM* ensures fairness by downgrading the DNN of the job(s) that have the highest share in the total resource utilization of the system, instead of directly changing the DNN of jobs that face SLA violation. Experimental results using both CPU- and GPU-based hardware platforms show that FairCIM can significantly improve the effective accuracy compared with approaches that are based on resource exchange. As part of future work, we can investigate the promise of combining resource exchange and DNN downgrading to further improve the latency and accuracy of latency-sensitive DNN inference jobs.

## Acknowledgment

This work was supported in part by NSF Grants #2105564 and #2236987, and a VMWare grant.

## REFERENCES


[1] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *ASPLOS*, 2019, pp. 107–120.

[2] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *IEEE HPCA*, 2020, pp. 193–206.

[3] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 681–696, 2016.

[4] W. Zhang, Q. Chen, N. Zheng, W. Cui, K. Fu, and M. Guo, "Toward qos-awareness and improved utilization of spatial multitasking gpus," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 866–879, 2022.

[5] G. Chen, Y. Zhao, X. Shen, and H. Zhou, "Effisha: A software framework for enabling effficient preemptive scheduling of gpu," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017, pp. 3–16.

[6] F. Guo, Y. Li, J. C. Lui, and Y. Xu, "Dcuda: Dynamic gpu schedulingith live migration support," in *ACM SoCC*, 2019, pp. 114–125.

[7] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference," in *IEEE/ACM ISCA*, 2020, pp. 982–995.

[8] J. Zhang, S. Elnikety, S. Zarar, A. Gupta, and S. Garg, "Model-switching: Dealing with fluctuating workloads in machine-learning-as-a-service systems," in *USENIX HotCloud*, 2020.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[10] G. Griffin, A. Holub, and P. Perona, "The caltech-256: Caltech technical report," *vol*, vol. 7694, p. 3, 2007.

[11] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[12] "Sentiment140," http://help.sentiment140.com/, 2022, accessed: April 25, 2022.

[13] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[14] L. Jiang, M. Xu, T. Liu, M. Qiao, and Z. Wang, "Deepvs: A deep learning based video saliency prediction approach," in *ECCV*, 2018, pp. 602–617.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015, pp. 1–9.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, and Z. Shlens, Jon andojna, "Rethinking the inception architecture for computer vision," in *CVPR'16*, 2016, pp. 2818–2826.

[18] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, e. ang, T. eyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE CVPR*, 2018, pp. 4510–4520.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE CVPR*, 2018, pp. 8697–8710.

[22] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018, pp. 19–34.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV'16*. Springer, 2016, pp. 630–645.

[24] W. Wang, J. Shen, F. Guo, M.-M. Cheng, and A. Borji, "Revisiting video saliency: A large-scale benchmark and a new model," in *IEEE CVPR*, 2018.

[25] W. Wang, J. Shen, J. Xie, M. Cheng, H. Ling, and A. Borji, "Revisiting video saliency prediction in the deep learning era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[26] S. M. Nabavinejad and T. Guo, "Opportunities of renewable energy powered dnn inference," *arXiv preprint arXiv:2306.12247*, 2023.

[27] F. Xu, J. Xu, J. Chen, L. Chen, R. Shang, Z. Zhou, and F. Liu, "igniter: Interference-aware gpu resource provisioning for predictable dnn inference in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 812–827, 2022.

[28] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[29] S. K. Ghosh, A. Raha, and V. Raghunathan, "Energy-efficient approximate edge inference systems," *ACM Transactions on Embedded Computing Systems*, 2023.

[30] S. S. Ogden and T. Guo, "Layercake: Efficient inference serving with cloud and mobile resources," in *IEEE/ACM CCGrid*, 2023, pp. 191–202.

[31] M. Wei, W. Zhao, Q. Chen, H. Dai, J. Leng, C. Li, W. Zheng, and M. Guo, "Predicting and reining in application-level slowdown on spatial multitasking gpus," *Journal of Parallel and Distributed Computing*, vol. 141, pp. 99–114, 2020.

[32] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant {GPU} clusters for {DNN} training workloads," in *USENIX ATC*, 2019, pp. 947–960.

[33] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *IEEE HPCA*, 2020, pp. 220–233.

[34] H. Zhao, W. Cui, Q. Chen, J. Leng, D. Zeng, and M. Guo, "Improving cluster utilization through adaptive resource management for dnn and cpu jobs co-location," *IEEE Transactions on Computers*, 2023.

[35] J. Mobin, A. Maurya, and M. M. Rafique, "Colti: Towards concurrent and co-located dnn training and inference," in *HPDC*, 2023, pp. 309–310.

[36] C. Wang, Y. Bai, and D. Sun, "Cd-msa: Cooperative and deadline-aware scheduling for efficient multi-tenancy on dnn accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[37] M. LeMay, S. Li, and T. Guo, "Perseus: Characterizing performance and cost of multi-tenant serving for cnn models," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2020, pp. 66–72.

[38] P. Jain, X. Mo, A. Jain, H. Subbaraj, R. S. Durrani, A. Tumanov, J. Gonzalez, and I. Stoica, "Dynamic space-time scheduling for gpu inference," *arXiv preprint arXiv:1901.00041*, 2018.

[39] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[40] W. Zhao, Q. Chen, H. Lin, J. Zhang, J. Leng, C. Li, W. Zheng, L. Li, and M. Guo, "Themis: Predicting and reining in application-level slowdown on spatial multitasking gpus," in *IEEE IPDPS*, 2019, pp. 653–663.

[41] N. Kulkarni, F. Qi, and C. Delimitrou, "Pliant: Leveraging approximation to improve datacenter resource efficiency," in *IEEE HPCA*, 2019, pp. 159–171.

[42] X. Wang, S. Chen, J. Setter, and J. F. Martínez, "Swap: Effective fine-grain management of shared last-level caches with minimum hardware support," in *IEEE HPCA*, 2017, pp. 121–132.

[43] Q. Chen, Z. Wang, J. Leng, C. Li, W. Zheng, and M. Guo, "Avalon: towards qos awareness and improved utilization through multi-resource management in datacenters," in *ACM ICS*, 2019, pp. 272–283.