iGniter: Interference-Aware GPU Resource Provisioning for Predictable DNN Inference in the Cloud

Fei Xu, *Member, IEEE*, Jianian Xu, Jiabin Chen, Li Chen, *Member, IEEE*, Ruitao Shang, Zhi Zhou, *Member, IEEE*, Fangming Liu, *Senior Member, IEEE*

Abstract—With the burgeoning demand for latency-sensitive artificial intelligence (AI)-based computation, GPUs are essential to accelerating deep neural network (DNN) inference in cloud datacenters. To fully utilize the GPU resources, *spatial sharing* of GPUs among co-located DNN inference workloads becomes increasingly compelling. Motivated by our empirical measurement study of DNN inference executed on EC2 GPU instances, we find that the performance interference among co-located inference workloads is noticeable, and we further identify the root cause of such interference as the *severe contention* of the GPU scheduler and GPU L2 cache space as well as the GPU power consumption. While existing works on guaranteeing performance SLOs of DNN inference focus on either *temporal sharing* of GPUs or *reactive* GPU resource scaling and inference migration techniques, how to *proactively* mitigate such severe performance interference has received comparatively little attention. In this paper, we propose *iGniter*, an *interference-aware* GPU resource provisioning framework for cost-efficiently achieving predictable DNN inference in the cloud. *iGniter* comprises of two key components: (1) a *lightweight* DNN inference performance model, which leverages the system and workload metrics that are practically accessible to explicitly capture the performance interference; (2) A *cost-efficient* GPU resource provisioning strategy that *jointly* optimizes the GPU resource allocation and adaptive batching based on our inference performance model, with the aim of achieving predictable performance of DNN inference workloads. We implement a prototype of *iGniter* based on NVIDIA Triton inference server on Amazon EC2 GPU instances. Extensive prototype experiments on four representative DNN models and datasets demonstrate that *iGniter* can guarantee the DNN inference performance SLOs, while saving the monetary cost by up to 25% in comparison to the state-of-the-art GPU resource provisioning strategies.

Index Terms—Cloud-based DNN inference, predictable performance, GPU resource provisioning, performance interference

1 Introduction

With the proliferating artificial intelligence (AI) applications, deep neural network (DNN) inference workloads are becoming increasingly commonplace in cloud datacenters [1]. While DNN models are getting more complex and thus consuming more computation and memory resources, GPUs have served as the *key* accelerator to reduce the inference latency and meet the service level objective (SLO) [2]. Hence, modern internet companies like Microsoft, Alibaba, and JD are increasingly adopting GPUs for serving DNN inference in their latency-critical products such as voice assistants [3], recommendation systems [4], and video analysis [5]. To cut down the inference budget and facilitate cloud-based DNN inference, most cloud providers have recently launched commercial cloud AI platforms such as AWS SageMaker [6] and Google Vertex AI [7]. As reported by Omdia [8], NVIDIA GPUs held an 80.6% market share of AI processors in cloud datacenters in 2020 and expect to reach 11.5 billion revenue worldwide by 2024 [9].

To improve the GPU resource utilization, *temporal sharing* [10] and *spatial sharing* [11] are two common GPU resource multiplexing techniques. There have been a number of works (*e.g.,* Cocktail [12], Clockwork [13]) on optimizing the DNN inference performance and monetary cost based on temporal sharing. However, a recent study [14] has shown that temporal sharing of GPUs to execute DNN inference workloads can intrinsically result in GPU resource wastage. To *fully exploit* the computation and memory ability of GPUs, NVIDIA has recently developed the multi-process service (MPS) [15] technique, which allows multiple inference workloads to *spatially* share the GPU resources with a limited percentage [16] (*e.g.,* 50%).

Though MPS can configure an amount of GPU execution resources for each inference workload, there exists *noticeable performance interference* among the *co-located* inference workloads on a GPU device, as evidenced by our motivation experiments in Sec. 2.2 that, the DNN inference latency can be prolonged by around 35% with only 5 co-located

- *Fei Xu, Jianian Xu, Jiabin Chen, Ruitao Shang are with the Shanghai Key Laboratory of Multidimensional Information Processing, School of Computer Science and Technology, East China Normal University, 3663 N. Zhongshan Road, Shanghai 200062, China. Email:* fxu@cs.ecnu.edu.cn.
- *Li Chen is with the School of Computing and Informatics, University of Louisiana at Lafayette, 301 East Lewis Street, Lafayette, LA 70504, USA. E-mail:* li.chen@louisiana.edu.
- *Zhi Zhou is with the Guangdong Key Laboratory of Big Data Analysis and Processing, School of Computer Science and Engineering, Sun Yat-sen University, 132 E. Waihuan Road, Guangzhou 510006, China. E-mail:* zhouzhi9@mail.sysu.edu.cn.
- *Fangming Liu is with the National Engineering Research Center for Big Data Technology and System, the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail:* fmliu@hust.edu.cn.

Manuscript received January XX, 2022; revised April XX, 2022.

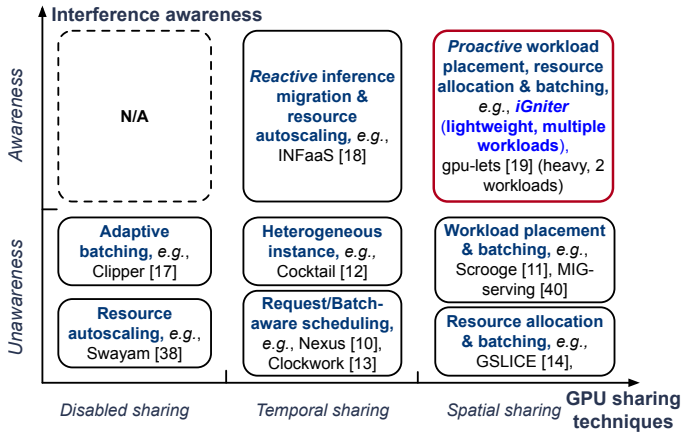

Fig. 1: The positioning of *iGniter* in the literature context of predictable DNN inference serving on GPUs.

workloads on a GPU. Such severe performance interference makes inference workloads easily suffer from unexpected SLO violations, which mainly originate from the shared *resource contention* in three aspects: (1) the *increased scheduling delay of kernels* by the GPU scheduler, and (2) the *severe contention of GPU L2 cache space*, as well as (3) the *reduced GPU frequency due to limited power cap*. Accordingly, it is essential to explicitly consider performance interference when provisioning GPU resources to DNN inference workloads, so as to meet the stringent performance SLOs for users.

As summarized in Fig. 1, there have recently been a number of research efforts devoted to guaranteeing the performance SLOs of DNN inference workloads, such as batch size configuration (*e.g.*, Clipper [17]), request scheduling (*e.g.*, Clockwork [13]), resource autoscaling (*e.g.*, Cocktail [12]), and GPU resource allocation (*e.g.*, GSLICE [14]). However, they are *oblivious* to the non-negligible performance interference among inference workloads, which is likely to cause resource under-provisioning and trigger *frequent reactive* adjustment of GPU resources, bringing heavy inference runtime overhead. There have also been several works on mitigating such performance interference through *reactive* inference migration (*e.g.*, INFaaS [18]) and characterizing the performance interference of *two* workloads using a linear regression model (*e.g.*, gpu-lets [19]). Nevertheless, such an interference model requires a large number (*i.e.*, thousands) of profiling for each workload and cannot readily be applied to multiple co-located inference workloads. As a result, there has been scant research attention paid to achieving predictable DNN inference by characterizing the performance inference in a *lightweight* manner and *proactively* mitigating such interference for inference workloads.

To fill this gap, in this paper, we design and implement *iGniter*, an *interference-aware* GPU resource provisioning framework to achieve predictable performance [20] (*i.e.*, latency and throughput) of DNN inference workloads while minimizing the inference budget in the cloud. To the best of our knowledge, *iGniter* is the first attempt to demonstrate how to *characterize the performance interference of DNN inference in a lightweight manner, and cost-efficiently provision GPU resources for inference workloads by jointly optimizing the GPU resource allocation and adaptive batching*. Specifically, we make the following contributions in *iGniter* as below.

▷ *First*, we build a *lightweight* analytical performance model to explicitly capture the performance interference among DNN inference workloads. It empirically leverages a set of essential system and workload metrics (*e.g.*, the GPU L2 cache utilization, the number of kernels) to characterize the severe contention of the GPU scheduler, the GPU L2 cache space, and the GPU power consumption as identified by our motivation experiments in Sec. 2.2.

▷ *Second*, we propose a *cost-efficient* GPU resource provisioning strategy to guarantee the performance SLOs of DNN inference workloads while reducing the inference budget. Given the DNN models with their latency and throughput SLOs, *iGniter jointly* optimizes the GPU resource allocation and adaptive batching based on our inference performance model. Specifically, it calculates the appropriate batch size and lower bound of allocated GPU resources that *just meet* the performance SLOs, and then it greedily identifies the GPU device for placement with the minimum performance interference for each inference workload.

▷ *Finally*, we implement a prototype[1] of *iGniter* based on NVIDIA Triton inference server [21] with three pieces of modules, including an *inference workload placer* and a *GPU resource allocator* as well as an *inference performance predictor*. We conduct the prototype experiments on a 10 V100 GPU cluster using 12 representative inference workloads of different latency SLOs and request arrival rates. Experiment results show that *iGniter* can provide predictable performance for DNN inference workloads, while reducing the monetary cost by up to 25%, compared with the state-of-the-art GPU resource provisioning strategies (*e.g.*, gpu-lets [19]).

The rest of the paper is organized as follows. Sec. 2 empirically analyzes the key factors that cause performance interference of inference workloads co-located on GPUs with MPS enabled, which motivates the design of our analytical performance model of DNN inference workloads in Sec. 3. Sec. 4 further designs and implements our *iGniter* GPU resource provisioning strategy for achieving predictable DNN inference serving in the cloud. Sec. 5 evaluates the effectiveness and runtime overhead of *iGniter*. Sec. 6 discusses related work and Sec. 7 concludes this paper.

2 Background and Motivation

In this section, we first seek to analyze the severity of performance inference among co-located DNN inference workloads and identify the key factors that cause such performance inference. Next, we present an illustrative example to show how to adequately provision the GPU resources to achieve predictable DNN inference.

2.1 Multi-Process Service of NVIDIA GPUs

To provide powerful computing ability, the NVIDIA GPU has been equipped with a number of Streaming Multiprocessors (SMs), and accordingly GPUs are currently widely used for hosting DNN inference workloads in the cloud [13]. To improve the resource utilization of GPUs, NVIDIA MPS [15] has been developed to share the GPU resources (*i.e.*, SMs) among multiple inference workloads executed on a single GPU device, and generally one process hosts one

1. https://github.com/icloud-ecnu/igniter

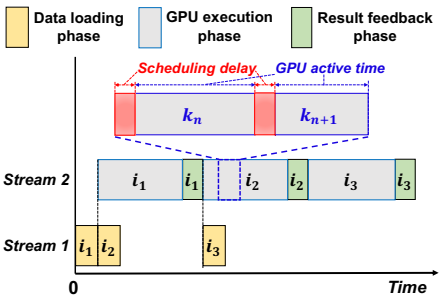

Fig. 2: The CUDA stream mechanism overlaps the execution of different DNN inference queries (*i.e.,* i_1, i_2, i_3) in an inference workload, and the kernels (*e.g.,* k_n) are scheduled to the SMs during the GPU execution phase.

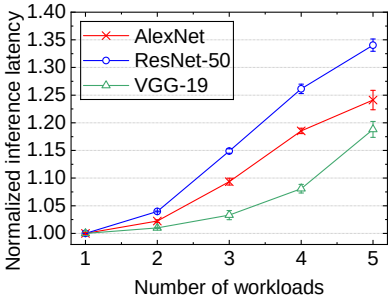

Fig. 3: The normalized inference latency of AlexNet, ResNet-50, and VGG-19 achieved on a V100 GPU, as the number of co-located inference workloads varies from 1 to 5, with respect to the workloads running alone.

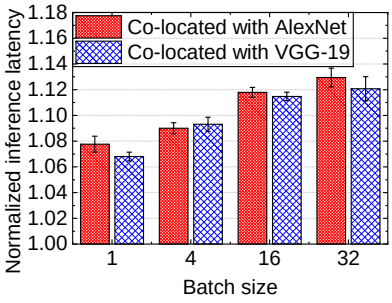

Fig. 4: The normalized inference latency of ResNet-50 when co-located with AlexNet or VGG-19 on a V100 GPU, as the batch size of AlexNet and VGG-19 varies from 1 to 32, with respect to ResNet-50 running alone.

inference workload. However, an uncontrollable allocation of GPU resources can degrade the Quality-of-Service (QoS) of DNN inference workloads. To deal with such a performance issue, MPS provisions each DNN inference workload with an amount of limited GPU resources (*i.e.,* a set of SMs), starting from the NVIDIA Volta architecture [16]. In general, the batch size of DNN inference also requires tuning to improve the GPU resource utilization, without violating the performance SLOs of inference workloads [2].

The execution of a DNN inference workload on a GPU device mainly has three phases: *First,* the host CPU transmits the inference input data to the GPU device over the PCIe interconnect. *Second,* the GPU device executes the DNN inference query. *Finally,* the inference result is transmitted back to the host CPU over the PCIe interconnect. To improve the GPU resource utilization, the mainstream DNN inference servers (*e.g.,* NVIDIA Triton [21]) have developed the CUDA *streams* to *overlap* the data loading phase and the GPU execution phase from different DNN inference queries in an *asynchronous* manner. As shown in Fig. 2, the DNN inference queries (*i.e.,* i_1, i_2, i_3) are launched in two different streams which can be executed concurrently. Specifically, Stream 1 (*i.e.,* the data loading phase of i_2 and i_3) overlaps with Stream 2 (*i.e.,* the GPU execution phase of i_1 and i_2). In particular, an inference query consists of a number of kernels (*e.g.,* k_n) which require *scheduling* to the SMs [22], leading to a moderate amount of *scheduling delay* of kernels in the GPU execution stream.

2.2 Performance Interference among Co-located DNN Inference Workloads

Though MPS facilitates the spatial GPU resource sharing among co-located DNN inference workloads, it inevitably brings *non-negligible* performance interference. To examine the *severity* of such performance interference, we conduct two motivation experiments using p3.2xlarge EC2 instances [23] equipped with NVIDIA V100 GPUs. Specifically, we *first* launch 1 to 5 identical inference workloads simultaneously and each workload is allocated with 20% of GPU resources. *Second,* we launch two DNN inference workloads on a GPU, which are allocated with 50% of GPU resources, respectively, and we vary the batch size of one workload from 1 to 32 while fixing the batch size as 16 for the other workload. We use AlexNet [24], ResNet-50 [25], and VGG-19 [26] models executed on the NVIDIA TensorRT [27] framework as our DNN inference workloads. In particular, the DNN inference latency is calculated by excluding the queueing delay. We illustrate the experimental results with error bars of standard deviation by repeating each experiment for three times.

As shown in Fig. 3 and Fig. 4, the DNN inference latency increases from 0.83% to 34.98% as the number of co-located workloads is varying from 2 to 5 and the batch size of co-located inference workloads is varying from 1 to 32. The experiment results indicate that the performance interference is *not uncommon* for MPS even with limited GPU resources (*i.e.,* GPU spatial sharing [15]). Our observation above is consistent with the findings in a more recent work [19]. Through an in-depth analysis, we find that such severe performance interference among DNN inference workloads is mainly caused by the following three factors.

Increased scheduling delay of kernels. Each kernel of a DNN inference workload is scheduled onto the SMs by the GPU scheduler. As shown in Fig. 5, we observe that: *First,* the scheduling delay shows a roughly linear increase as the number of co-located workloads increases from 2 to 5. We conjecture that the GPU scheduler requires scheduling the kernels from different inference workloads onto the SMs in a round-robin manner. *Second,* the scheduling delay of ResNet-50 increases much faster than AlexNet. This is simply because the number of kernels of ResNet-50 is bigger than that of AlexNet.

Severe contention of GPU L2 cache space. Though MPS is able to partition the GPU resources, the GPU L2 cache space is still shared by the co-located DNN inference workloads [28]. To characterize the severity of L2 cache contention on a GPU device, we simply adopt a system metric, *i.e.,* the L2 cache request hit ratio. As shown in Fig. 6, we observe that the GPU active time (*i.e.,* GPU execution latency - GPU scheduling delay, as depicted in Fig. 2) of an inference workload (*e.g.,* ResNet-50) is highly affected the contention of GPU L2 cache space. As the number of co-located workloads increases, the severer cache contention leads to a smaller L2 cache hit ratio, which in turn increases the GPU active time of an inference workload.

Reduced GPU frequency due to limited power cap. Reduction of GPU frequency brings performance degradation to GPU workloads [29]. As shown in Fig. 7, we observe that: *First,* the GPU frequency starts to decrease once the GPU power reaches its upper limit value. This is because more inference workloads consume a larger amount of power on a GPU

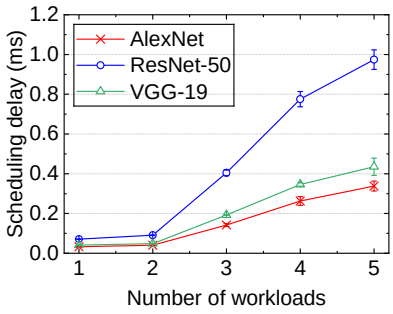

Fig. 5: The scheduling delay of AlexNet, ResNet-50, and VGG-19 with different numbers of workloads executed on a V100 GPU.

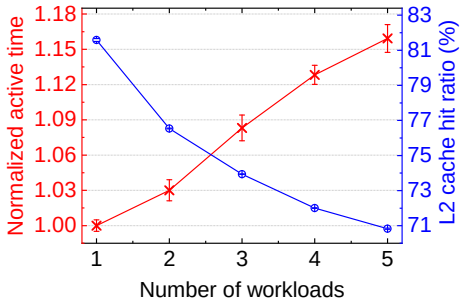

Fig. 6: The GPU active time and L2 cache request hit ratio of ResNet-50 with different numbers of workloads executed on a V100 GPU.

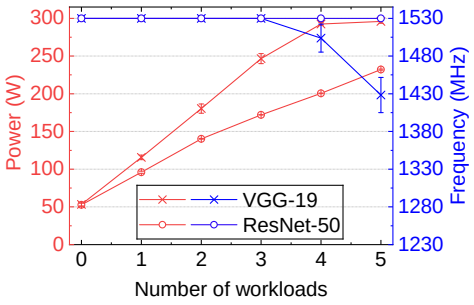

Fig. 7: The GPU power and GPU frequency of VGG-19 and ResNet-50 with different numbers of workloads executed on a V100 GPU.

TABLE 1: Comparison of GPU resource provisioning plans and SLO violations achieved by the gpu-lets, GSLICE and our *iGniter* strategies for three representative DNN models (*i.e.*, AlexNet (A), ResNet-50 (R), VGG-19 (V)).

Approaches	Resource provisioning plans GPU: model(#resource, #batch)	Violations
GSLICE [14]	GPU1 : A(37.5%, 18), R(30%, 8), V(40%, 6)	2 models (A, R)
gpu-lets [19]	GPU1 : A(40%, 23) GPU2 : R(60%, 18), V(40%, 6)	2 models (A, R)
iGniter	GPU1 : A(10%, 4), R(30%, 8), V(37.5%, 6)	None

device, while the GPU has to maintain the upper limit of GPU power through the frequency reduction. *Second,* the GPU power of VGG-19 and ResNet-50 shows a roughly linear relationship to the number of inference workloads, as long as the GPU power is below its upper limit value.

Based on our analysis above, we further explain why the batch size of co-located workloads (*e.g.,* AlexNet, VGG-19) can moderately affect the DNN inference performance (*e.g.,* ResNet-50), as shown in Fig. 4. Such performance interference can be attributed to the resource contention of the GPU L2 cache space and GPU power. Specifically, the GPU L2 cache utilization of AlexNet and VGG-19 increases from 11.1% to 18.4% and from 16.9% to 22.0%, respectively, while the GPU power of the two workloads increases from 108 W to 156 W and from 139 W to 179 W, respectively, as the batch size increases from 1 to 32. Accordingly, such severe contention of the GPU L2 cache space and GPU power from co-located inference workloads inevitably prolongs the DNN inference latency.

Summary: *First,* the performance interference among DNN inference workloads cannot be overlooked. We identify the main factors that cause such performance interference as the severe contention of the GPU scheduler, the GPU L2 cache space, and the GPU power consumption among the co-located inference workloads. *Second,* explicitly considering the performance interference is compelling when provisioning GPU resources to DNN inference workloads, so as to guarantee the performance of DNN inference workloads.

2.3 An Illustrative Example

To achieve predictable DNN inference performance (*i.e.,* latency and throughput) and cost-efficient GPU resource provisioning, we propose *iGniter* in Sec. 4 and illustrate its effectiveness by conducting another motivation experiment with the AlexNet, ResNet-50 and VGG-19 models. We set the latency SLOs (ms) and request arrival rates (req/s) for the three inference workloads as 15, 40, 60 and 500, 400, 200, respectively. We define the P99 latency of an inference workload exceeding its latency SLO as a violation, while each resource provisioning strategy aims at limiting the average inference latency of each workload below half of its SLO by taking account of the batching latency [10].

As shown in Table 1, GSLICE [14] and gpu-lets [19] require 1 GPU and 2 GPUs, respectively, and unfortunately make two DNN models violate their SLOs. Our *iGniter* strategy provisions 1 GPU for hosting the three models appropriately and guarantees their SLOs. Specifically, we find that GSLICE and gpu-lets tend to provision more GPU resources and larger batch sizes to AlexNet and ResNet-50. This is because the two strategies aim to maximize the throughput while guaranteeing the latency SLOs. In more detail, GSLICE [14] is an *interference-unaware* strategy, which tunes the allocated GPU resources and batch size *separately*, and accordingly, the total allocated resources can exceed the maximum resources (*i.e.,* 100%) of a GPU device which inevitably leads to the contention of SMs and thus a high long-tail latency.

Though gpu-lets [19] explicitly considers the performance interference, it works *only for two inference workloads* on a GPU device. Also, gpu-lets only considers the performance interference for the newly-placed inference workload (*i.e.,* VGG-19), and it does not change the allocated GPU resources and batch size of the workload (*i.e.,* ResNet-50) that has already been placed on the GPU. Accordingly, the inference latency of ResNet-50 exceeds its latency SLO due to the co-located interference of VGG-19. Moreover, gpu-lets first provisions GPU resources as the most efficient amount of GPU resources and then sets the batch sizes as large as possible. However, a large batch size cannot fully utilize the GPU resources at low request arrival rate and it can cause SLO violations due to a long batching latency. In contrast, *iGniter* sets an appropriate batch size for inference workloads that *just meet* their latency SLO and request arrival rate, and it further provisions GPU resources by explicitly considering the performance interference among multiple (more than 2) inference workloads. Accordingly, our *iGniter* strategy can guarantee the performance of DNN inference workloads in a cost-efficient manner.

TABLE 2: Key notations in our DNN inference performance model.

Notation	Definition
$\mathcal{I}, \mathcal{J}$	Sets of DNN inference workloads and allocated GPUs
t_{inf}^{ij}	DNN inference latency of an inference workload i on a GPU j
h^{ij}	Throughput of an inference workload i on a GPU j
t_{load}^{i}, $t_{feedback}^{i}$	DNN inference data loading latency and result feedback latency of an inference workload i
t_{gpu}^{ij}	GPU execution latency of an inference workload i on a GPU j
t_{sch}^{ij}, t_{act}^{ij}	Scheduling delay and GPU active time of an inference workload i on a GPU j
f^j	Actual frequency of a GPU j
p_{demand}^{j}	Total power demand of a GPU j
k_{act}^{i}	GPU active time of an inference workload i when running alone on a GPU device
p^i, c^i	Power consumption and L2 cache utilization of an inference workload i when running alone on a GPU device
r^{ij}, v^{ij}	GPU resource allocation and placement of an inference workload i on a GPU j
b^i	Batch size of an inference workload i

3 Modeling DNN Inference Performance on GPUs

In this section, we first build an analytical model to predict the DNN inference performance (*i.e.*, latency and throughput) in the cloud. We explicitly consider the performance interference among DNN inference workloads with different batch sizes and allocated GPU resources. We next formulate the GPU resource provisioning problem to minimize the monetary cost while guaranteeing the inference latency and request arrival rate. The key notations in our performance model are summarized in Table 2.

3.1 Predicting DNN Inference Performance with GPU Resources

We consider a set of constantly-arrived DNN inference workloads denoted by $\mathcal{I} = \{i_1, i_2, ..., i_m\}$ and a set of allocated GPU devices denoted by $\mathcal{J} = \{j_1, j_2, ..., j_g\}$ with *a given GPU type*. As elaborated in Sec. 2.1, the execution of DNN inference on the GPU can be divided into three sequential steps: *data loading*, *GPU execution*, and *result feedback*. Accordingly, the DNN inference latency t_{inf}^{ij} of a workload i executed on a GPU device j can be calculated by summing up the data loading latency t_{load}^{i}, the GPU execution latency t_{gpu}^{ij}, and the result feedback latency $t_{feedback}^{i}$, which is formulated as

$$t_{inf}^{ij} = t_{load}^{i} + t_{gpu}^{ij} + t_{feedback}^{i}. \tag{1}$$

To improve the GPU resource utilization, the data loading phase overlaps with the GPU execution and result feedback phases in the mainstream DNN inference servers (*e.g.*, Triton [21]), as shown in Fig. 2. Accordingly, we estimate the DNN inference throughput h^{ij} as

$$h^{ij} = \frac{b^i}{t_{gpu}^{ij} + t_{feedback}^{i}}, \tag{2}$$

where $b^i \in \mathcal{N}^+$ denotes the batch size of an inference workload $i \in \mathcal{I}$.

Data loading and result feedback phases. As discussed in Sec. 2.1, the inference input and result data are transmitted between the CPU and GPU devices via the PCIe. Basically, the sizes of inference input data and result data are both linear to the batch size b^i. We calculate the data loading latency t_{load}^{i} and the result feedback latency $t_{feedback}^{i}$ as

$$t_{load}^{i} = \frac{d_{load}^{i} \cdot b^i}{B_{pcie}} \quad \text{and} \quad t_{feedback}^{i} = \frac{d_{feedback}^{i} \cdot b^i}{B_{pcie}}, \tag{3}$$

respectively, where d_{load}^{i} and $d_{feedback}^{i}$ are the data sizes of input data and result data, respectively, when $b^i = 1$. B_{pcie} denotes the available PCIe bandwidth of the given GPU type.

GPU execution phase. Each DNN inference workload is executed with an amount of allocated GPU resources denoted by $r^{ij} \in [0, r_{max}], \forall i \in \mathcal{I}, j \in \mathcal{J}$, which are actually mapped to a set of SMs [15]. In general, r_{max} is set as 1. As depicted in Fig. 2, the GPU execution phase consists of the GPU scheduling delay and kernels running on the allocated SMs, which is directly affected by the allocated GPU resources r^{ij}. Furthermore, the performance interference can be caused by the reduction of GPU frequency due to the workload co-location, which inevitably prolongs the GPU execution phase, as evidenced by Sec. 2.2. Accordingly, we formulate the GPU execution latency t_{gpu}^{ij} as

$$t_{gpu}^{ij} = \frac{t_{sch}^{ij} + t_{act}^{ij}}{\frac{f^j}{F}}, \tag{4}$$

where t_{sch}^{ij} and t_{act}^{ij} denote the total scheduling delay of kernels and the GPU active time of an inference workload i executed on a GPU device j, respectively, *without any GPU frequency reduction*. f^j and F denote the actual and the maximum GPU frequency, respectively, on a GPU device j.

In the following, we first model the *scheduling delay* t_{sch}^{ij} of DNN inference workloads. Intuitively, t_{sch}^{ij} is roughly linear to the number of kernels n_k^i for a DNN inference workload i, which can be estimated as

$$t_{sch}^{ij} = (k_{sch}^{i} + \Delta_{sch}^{j}) \cdot n_k^i, \tag{5}$$

where n_k^i denotes the number of kernels of an inference workload i, and k_{sch}^{i} denotes the scheduling delay when the workload i running alone on a GPU device. Δ_{sch}^{j} is the increased scheduling delay caused by the performance interference on the GPU resource scheduler, which is highly relevant to the number of co-located inference workloads as evidenced by Sec. 2.2. Accordingly, we estimate the increased scheduling delay as

$$\Delta_{sch}^{j} = \begin{cases} 0 & \sum_{i \in \mathcal{I}} v^{ij} \leq 1, \\ \alpha_{sch} \cdot \sum_{i \in \mathcal{I}} v^{ij} + \beta_{sch} & \text{otherwise.} \end{cases} \tag{6}$$

where α_{sch} and β_{sch} are the coefficients to characterize the increased scheduling delay on a given GPU type. $\sum_{i \in \mathcal{I}} v^{ij}$ denotes the number of co-located inference workloads on a GPU device j. In particular, v^{ij} denotes whether an inference workload i is running on a GPU device j, which can be given by

$$v^{ij} = \begin{cases} 1 & \text{a workload } i \text{ runs on a GPU } j, r^{ij} > 0, \\ 0 & \text{otherwise, } r^{ij} = 0. \end{cases} \tag{7}$$

We next model the *GPU active time* t_{act}^{ij} of an inference workload i executed on a GPU device j. Due to the contention on the shared GPU L2 cache space, the GPU active time is inversely proportional to GPU L2 cache hit ratio on a GPU device as evidenced by Sec. 2.2. We simply leverage a system-level metric called *GPU L2 cache utilization* to characterize the workload *demand* on the GPU L2 cache space. Given a fixed *supply* of L2 cache space on a GPU device, a higher GPU L2 cache utilization (*i.e.*, *demand*) indicates severer contention on the GPU L2 cache space, thereby resulting in a longer GPU active time. Accordingly, we estimate the GPU active time t_{act}^{ij} as

$$t_{act}^{ij} = k_{act}^{i} \cdot \Big(1 + \alpha_{cache}^{i} \cdot \sum_{i \in \mathcal{I} \setminus i} \big(c^{i} \cdot v^{ij}\big)\Big), \tag{8}$$

where α_{cache}^{i} denotes the coefficient to characterize the prolonged GPU active time due to L2 cache contention for an inference workload i. k_{act}^{i} and c^{i} are the GPU active time and L2 cache utilization, respectively, when an inference workload i running alone on a given GPU type.

Finally, we model the *GPU frequency* f^{j} on a GPU device j. As evidenced by Sec. 2.2, the GPU frequency decreases dramatically as the total GPU power demand p_{demand}^{j} of workloads exceeds the upper limit of GPU power supply P of the given GPU type. As the GPU frequency is highly relevant to the GPU power [29], we estimate the GPU frequency as

$$f^{j} = \begin{cases} F & p_{demand}^{j} \le P, \\ F - \alpha_{f} \cdot (p_{demand}^{j} - P) & p_{demand}^{j} > P, \end{cases} \tag{9}$$

where α_{f} denotes the coefficient to characterize the relationship between the GPU power and frequency on a given GPU type. In addition, we estimate the total power demand of a GPU device j by summing up the power consumption p^{ij} of inference workloads and the *idle* power p_{idle} of the given GPU type, which is given by

$$p_{demand}^{j} = p_{idle} + \sum_{i \in \mathcal{I}} \big(p^{i} \cdot v^{ij}\big). \tag{10}$$

In particular, we obtain p^{i} by running an inference workload i alone on a GPU device of the given type.

Obtaining model coefficients: Based on the above, we have 8 *workload-specific* coefficients (*i.e.*, d_{load}^{i}, $d_{feedback}^{i}$, n_{k}^{i}, k_{sch}^{i}, k_{act}^{i}, p^{i}, c^{i}, α_{cache}^{i}) and 7 *hardware-specific* coefficients (*i.e.*, P, F, p_{idle}, B_{pcie}, α_{f}, α_{sch}, β_{sch}) in our performance model. Specifically, four workload-specific coefficients (*i.e.*, d_{load}^{i}, $d_{feedback}^{i}$, n_{k}^{i}, k_{sch}^{i}) can be obtained by profiling the workload *only once* using the `Nsight Systems` [30]. Given a GPU type, three hardware-specific coefficients (*i.e.*, P, F, p_{idle}) can be obtained using the $\texttt{nvidia} - \texttt{smi}$ [31]. The available PCIe bandwidth B_{pcie} can be obtained by transferring data from CPU memory to GPU memory. The GPU frequency coefficient α_{f} and scheduling coefficients α_{sch} and β_{sch} can be obtained by launching 1 to 5 inference workloads concurrently on a given GPU type. The cache coefficient α_{cache}^{i} can be obtained by launching multiple (*i.e.*, 2) inference workloads concurrently. Moreover, we obtain the GPU active time k_{act}^{i}, power consumption p^{i}, and the L2 cache utilization c^{i} for an inference workload i running alone on a given GPU type. In particular, k_{act}^{i}, p^{i} and c^{i} are affected by different batch sizes b^{i} and allocated GPU resources r^{ij}.

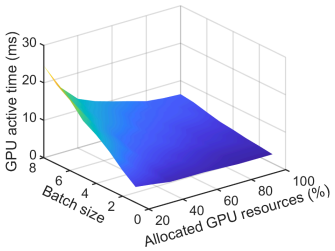

Fig. 8: The GPU active time of ResNet-50 with different batch sizes and allocated GPU resources.

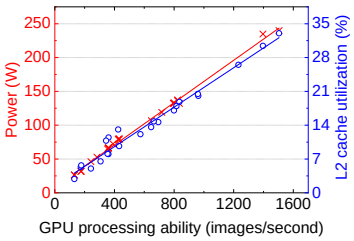

Fig. 9: The power consumption and L2 cache utilization of ResNet-50 with different GPU processing abilities.

We first model the GPU active time k_{act}^{i} in terms of the allocated GPU resources r^{ij} and the batch size b^{i}. As shown in Fig. 8, the GPU active time shows a roughly inverse proportion to the amount of allocated GPU resources. Meanwhile, the GPU active time increases , which can be formulated by a quadratic function. Accordingly, we formulate the GPU active time k_{act}^{i} as

$$k_{act}^{i} = \frac{k_{1}^{i} \cdot (b^{i})^{2} + k_{2}^{i} \cdot b^{i} + k_{3}^{i}}{r^{ij} + k_{4}^{i}} + k_{5}^{i}, \tag{11}$$

where k_{1}^{i}, k_{2}^{i}, k_{3}^{i}, k_{4}^{i}, k_{5}^{i} denote the model coefficients of an inference workload i. We first profile such an inference workload with several (*i.e.*, 9) different batch sizes and allocated GPU resources. Then, we obtain the model coefficients by fitting the profiled data using the *least squares method* [32].

We next find the relationship between power consumption p^{i}, L2 cache utilization c^{i} and GPU processing ability (*i.e.*, $\frac{b}{k_{act}}$). Specifically, we collect the power consumption and L2 cache utilization of ResNet-50 with different GPU processing abilities. As shown in Fig. 9, we observe that both the power consumption and L2 cache utilization grow *linearly* with the GPU processing ability. This is because a stronger GPU processing ability leads to a higher GPU resource utilization and power consumption. Accordingly, the relationship between power consumption p^{i}, L2 cache utilization c^{i} and GPU processing ability $\frac{b}{k_{act}}$ can be estimated as

$$p^{i} = \alpha_{power}^{i} \cdot \frac{b^{i}}{k_{act}^{i}} + \beta_{power}^{i},$$
$$c^{i} = \alpha_{cache}^{i} \cdot \frac{b^{i}}{k_{act}^{i}} + \beta_{cache}^{i},$$

where α_{power}^{i}, β_{power}^{i} and α_{cache}^{i}, β_{cache}^{i} denote the coefficients to characterize the relationship between the power consumption, L2 cache utilization and the GPU processing ability. Similarly, we obtain these model coefficients by fitting our profiled workload data using the *least squares method* [32]. The L2 cache utilization can be obtained using `Nsight Compute` [33]. In particular, we only require profiling each inference workload with 11 different configurations of allocated GPU resources and batch size, which is far less than the number (*e.g.*, $40 \times 32 = 1,280$) of all possible configurations of allocated GPU resources (*e.g.*, 40 choices) and batch sizes (*e.g.*, 32 choices) for each inference workload, even without considering performance interference.

3.2 Analyzing GPU Resource Provisioning Optimization Problem

Based on our DNN inference performance model above, we proceed to define the optimization problem of GPU resource provisioning for DNN inference workloads as follows: *Given the request arrival rate R^i and the latency SLO T^i_{slo}, how can we provision the GPU resource r^{ij} and configure the batch size b^i for each inference workload i, in order to achieve predictable DNN inference performance while minimizing the monetary cost C of GPU resource provisioning?* Accordingly, our optimization problem can be formulated as

$$\min_{b^i, r^{ij}} \quad C = \sum_{j \in \mathcal{J}} u^j \tag{12}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} h^{ij} \cdot v^{ij} \geq R^i, \quad \forall i \in \mathcal{I} \tag{13}$$

$$\sum_{j \in \mathcal{J}} t^{ij}_{inf} \cdot v^{ij} \leq \frac{T^i_{slo}}{2}, \quad \forall i \in \mathcal{I} \tag{14}$$

$$\sum_{i \in \mathcal{I}} r^{ij} \leq r_{max}, \quad \forall j \in \mathcal{J} \tag{15}$$

$$\sum_{j \in \mathcal{J}} v^{ij} = 1, \quad \forall i \in \mathcal{I} \tag{16}$$

where u^j denotes the unit price for each GPU device j, and Eq. (12) defines our objective function which minimizes the monetary cost C of GPU resource provisioning, subject to the following four constraints. Specifically, Constraint (13) guarantees that the throughput of each inference workload can meet its arrival rate. Constraint (14) guarantees the inference latency of each inference workload below its objective latency $\frac{T^i_{slo}}{2}$. This is because the batch inference latency cannot exceed half of the SLO [10] by taking account of the performance impact of request batching and queueing. Constraint (15) denotes that the allocated GPU execution resources of each GPU device should be no more than the maximum GPU execution resources r_{max}. Constraint (16) denotes that each inference workload can only be placed on one GPU device.

Problem analysis: According to Eq. (12), the monetary cost C is affected by the unit price u^j and the set of allocated GPU devices $\mathcal{J}$. As u^j becomes a constant value u given a GPU type, the optimization problem can be reduced to minimizing the number $|\mathcal{J}|$ of allocated GPU devices. To achieve such a goal, each inference workload requires to be allocated GPU resources that *just meet* the request arrival rate and latency SLOs.

Theorem 1. *Given a DNN inference workload with the arrival rate and SLO, the lower bound r^i_{lower} of allocated GPU resources* (i.e., *the allocated GPU resources that DNN inference workloads are running alone on a GPU device) and the appropriate batch size b^i_{appr} can be calculated as*

$$b^i_{appr} = \left\lceil \frac{T^i_{slo} \cdot R^i \cdot B_{pcie}}{2 \cdot (B_{pcie} + R^i \cdot d^i_{load})} \right\rceil, \tag{17}$$

$$r^i_{lower} = \left\lceil \frac{\gamma^i}{\delta^i \cdot r_{unit}} - \frac{k^i_4}{r_{unit}} \right\rceil \cdot r_{unit}. \tag{18}$$

where $\gamma^i = k^i_1 \cdot (b^i_{appr})^2 + k^i_2 \cdot b^i_{appr} + k^i_3$ and $\delta^i = \frac{T^i_{slo}}{2} - \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch}$. r_{unit} denotes the allocation unit of GPU resources, which can be empirically set as 2.5% (i.e., around 2 SMs) for NVIDIA V100 GPUs.

The proof can be found in Appendix A. Our selected *appropriate* batch size b^i_{appr} can guarantee the request arrival rate by letting $t^{ij}_{gpu} = \frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback}$. Accordingly, Constraint (13) and Constraint (14) can be combined into one constraint, and thus the original optimization problem in Eq. (12) can be simplified as

$$\min_{r^{ij}} \quad \frac{u}{r_{max}} \cdot \left(\sum_{i \in \mathcal{I}} r^i_{lower} + \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} r^{ij}_{inter} + \sum_{j \in \mathcal{J}} r^j_f \right)$$

$$\text{s.t.} \quad \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} + \sum_{j \in \mathcal{J}} t^{ij}_{gpu} \leq \frac{T^i_{slo}}{2}, \ \forall i \in \mathcal{I}$$

(15), (16),

where $r^{ij}_{inter} = r^{ij} - r^i_{lower} \cdot v^{ij}$ is the *increased* GPU resources caused by the performance interference of co-located inference workloads. $r^j_f = r_{max} - \sum_{i \in \mathcal{I}} r^{ij}$ denotes the *unallocated* GPU resource fragments on a GPU device j. Accordingly, given the fixed lower bound r^i_{lower} of GPU resources, our optimization problem can be transformed into minimizing the *GPU resource fragmentation* and the *increased GPU resources* caused by the performance interference. Suppose that there is no performance interference among the inference workloads (*i.e.*, $r^{ij}_{inter} = 0$), our problem can be reduced to a classic *bin packing problem* which is already shown to be NP-hard [34]. Obviously, our original optimization problem is more *complicated* than such a bin packing problem. Accordingly, we turn to devising a heuristic algorithm to acquire a sub-optimal solution to our GPU resource provisioning problem.

4 Design of *iGniter*: Guaranteeing Performance of DNN Inference Workloads

Based on our analysis of DNN inference performance model on GPU devices and our optimization problem defined in Sec. 3, we further present *iGniter* in Alg. 1, a *simple yet effective* GPU resource provisioning strategy to solve our optimization problem. Our *iGniter* strategy aims to provide predictable performance (*i.e.*, guarantee the SLO latency and request arrival rate) for DNN inference workloads, while minimizing the amount of provisioned GPU resources (and thus the monetary cost) in the cloud.

4.1 Algorithm Design

To particularly answer "*how to provision GPU resources for DNN inference workloads*," our *iGniter* strategy in Alg. 1 is quite *intuitive*: We first decide *where to place* inference workloads and then identify *how to allocate* GPU resources to the workloads. To reduce the unallocated GPU resource fragments, *iGniter* sorts the inference workloads according to r^i_{lower} in a *descending* order and puts them onto a new GPU device *only when* there are not enough GPU resources, which is in accordance with the `ANYFIT` constraint [34]. To greedily reduce the increased GPU resources caused by the interference, *iGniter* judiciously selects an appropriate GPU device with the *least* performance interference to host each inference workload.

Inference workload placement strategy: Given a set of DNN inference workloads with their latency SLOs T^i_{slo} and request arrival rates R^i, *iGniter* first obtains the *hardware-specific* coefficients (*i.e.*, P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch}) and the *workload-specific* coefficients (*i.e.*, d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache}) for each inference workload through a *lightweight* coefficient acquisition method elaborated in Sec. 3.1 (line 1). With these obtained coefficients, *iGniter* calculates the appropriate batch size b^i_{appr} by Eq. (17) and the lower bound of GPU resources r^i_{lower} by Eq. (18), and then initializes the provisioned GPU resources r^{ij} and the number of allocated GPUs g (line 2). By iterating over the *sorted* inference workloads set $\mathcal{I}$, *iGniter* greedily finds an *appropriate* GPU device for each workload (lines 3-12). In more detail, *iGniter* initializes the allocated GPU resources r^{ij}_a after placing the inference workload on the GPU, and the minimum increased GPU resources r^{min}_{inter} caused by the performance interference for placing the workload (lines 5). For each candidate GPU, *iGniter* first calculates the allocated GPU resources r^{ij}_a and the increased resources r^{ij}_{inter} by Alg. 2 (lines 6-8), and then it greedily identifies the *appropriate* GPU q which can host the inference workload and cause the least performance interference (lines 9-12). Finally, *iGniter* provisions a new GPU device if there are no enough resources for the inference workload w (*i.e.*, $q == -1$), and then places such a workload w onto the GPU device q with the minimum increased GPU resources (lines 15-18).

GPU resource allocation strategy: `alloc_gpus` first initializes the allocated GPU resources r^{wj}_a of the workload w on the GPU j (line 1). `alloc_gpus` then iteratively *reallocates* the GPU resources for each workload i on the GPU j, as long as SLO violations occur for an inference workload i and the GPU j still has enough unallocated GPU resources (lines 2-11). Specifically, `alloc_gpus` calculates the inference latency t^{ij}_{inf} by Eq. (1) and judges whether the SLO violation occurs for each workload i (lines 4-6). For these SLO violated workloads, `alloc_gpus` increases the allocated GPU resources by a unit of GPU resources (*i.e.*, r_{unit}) in order to guarantee the inference SLOs (lines 7-11).

Remark: As Alg. 1 (line 7) invokes Alg. 2, we analyze that the complexity of Alg. 1 can be in the order of $\mathcal{O}(m \cdot g \cdot n \cdot \frac{m}{g})$, where m denotes the number of inference workloads and g denotes the number of allocated GPUs. $n = \frac{r_{max} - \sum_{i \in \mathcal{I}} r^{ij}_a}{r_{unit}} + 1$ denotes the cardinality of searching space of the allocated GPU resources for an inference workload, and $\frac{m}{g}$ denotes the *expected* number of inference workloads co-located on a GPU. Accordingly, the complexity of Alg. 1 can be reduced to $\mathcal{O}(m^2)$ as n is limited (*i.e.*, at most 40 values). The runtime overhead of our *iGniter* strategy is well contained and will be validated in Sec. 5.4.

4.2 Implementation of *iGniter*

We implement a prototype of *iGniter* framework running on AWS EC2 [23] based on NVIDIA Triton [21], which is a representative cloud inference server. In more detail, our prototype of *iGniter* is built upon Triton v2.12.0 supported by the TensorRT v8.0.1.6 backend framework, with over $1,000$ lines of Python, C++, and Linux Shell codes. The source codes of our *iGniter* prototype are publicly available on GitHub (*i.e.*, https://github.com/icloud-ecnu/igniter).

Algorithm 1: *iGniter:* Cost-efficient GPU resource provisioning strategy for predictable performance of DNN inference workloads.

Input: The latency SLO T^i_{slo} and the request arrival rate R^i for each inference workload $i \in \mathcal{I}$.
Output: Cost-efficient resource provisioning plan, including the provisioned GPU resources r^{ij} and the appropriate batch size b^i_{appr} as well as the number of allocated GPUs g.
1: Acquire *hardware-specific* coefficients P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch} for a given GPU type, and obtain *workload-specific* coefficients d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache} through profiling each workload $i \in \mathcal{I}$;
2: **Initialize:** the appropriate batch size $b^i_{appr} \leftarrow$ Eq. (17), the lower bound of GPU resources $r^i_{lower} \leftarrow$ Eq. (18), and $r^{ij} \leftarrow 0, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}$, as well as $g \leftarrow 1$;
3: Sort workloads according to r^i_{lower} in a descending order;
4: **for all** workload w in $\mathcal{I}$ to be placed on GPUs **do**
5: **Initialize:** the allocated GPU resources $r^{ij}_a \leftarrow r^{ij}$, $\forall i \in \mathcal{I}, \forall j \in \mathcal{J}$, after placing an inference workload w on the GPU $q \leftarrow -1$, and the minimum increased GPU resources caused by the *performance interference* $r^{min}_{inter} \leftarrow r_{max}$, for placing the workload w;
6: **for all** GPU device j in $[1, g]$ **do**
7: $r^{ij}_a \leftarrow$ `alloc_gpus`$(T^i_{slo}, r^{ij}_a, r^w_{lower})$;
8: Calculate the increased GPU resources caused by the *performance interference* $r^{ij}_{inter} \leftarrow r^{ij}_a - r^{ij}, \forall i \in \mathcal{I}$ on the GPU j;
9: **if** $\left(\sum_{i \in \mathcal{I}} r^{ij}_a \leq r_{max}\right)$ && $\left(\sum_{i \in \mathcal{I}} r^{ij}_{inter} < r^{min}_{inter}\right)$ **then**
10: Set $q \leftarrow j$, and $r^{min}_{inter} \leftarrow \sum_{i \in \mathcal{I}} r^{ij}_{inter}$;
11: **end if**
12: **end for**; // find an appropriate GPU for a workload w
13: **if** $q == -1$ **then**
14: Update $g \leftarrow g + 1$, and $r^{wg} \leftarrow r^w_{lower}$; // add one GPU
15: **else**
16: Update $r^{iq} \leftarrow r^{iq}_a, \forall i \in \mathcal{I}$; // enough GPU resources
17: **end if**
18: **end for**

Algorithm 2: `alloc_gpus`: GPU resource allocation algorithm for placing an inference workload on a GPU device.

Input: The latency SLO T^i_{slo} and the allocated GPU resources r^{ij}_a for each inference workload $i \in \mathcal{I}$, before placing the inference workload w on the GPU j, as well as the resource lower bound r^w_{lower} of the inference workload w.
Output: Allocated GPU resources r^{ij}_a, after placing the inference workload w on the GPU j.
1: **Initialize:** the allocated GPU resources $r^{wj}_a \leftarrow r^w_{lower}$ of the workload w on the GPU j, and whether the GPU resources require reallocation $flag \leftarrow 1$;
2: **while** $\left(\sum_{i \in \mathcal{I}} r^{ij}_a \leq r_{max}\right)$ && $\left(flag == 1\right)$ **do**
3: **Initialize:** $flag \leftarrow 0$;
4: **for all** inference workload i on the GPU j **do**
5: Calculate the inference latency $t^{ij}_{inf} \leftarrow$ Eq. (1);
6: **if** $t^{ij}_{inf} > \frac{T^i_{slo}}{2}$ **then**
7: Increase the allocated GPU resources $r^{ij}_a \leftarrow r^{ij}_a + r_{unit}$ for a workload i;
8: Set $flag \leftarrow 1$;
9: **end if**; // SLO violation occurs
10: **end for**; // Reallocate GPU resources
11: **end while**

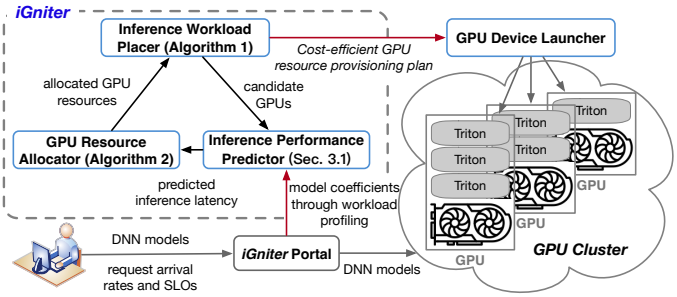

Fig. 10: The prototype of *iGniter* on AWS EC2.

As illustrated in Fig. 10, our *iGniter* framework comprises three pieces of modules: an *inference workload placer* and a *GPU resource allocator* as well as an *inference performance predictor*. Specifically, users submit the DNN models and their request arrival rates and SLOs to the *iGniter portal*, which can be deployed on a low-end EC2 instance and initiates a *lightweight* workload profiling on *different types* of GPU devices to acquire the model coefficients (*i.e.*, workload-specific and hardware-specific coefficients as elaborated in Sec. 3.1). With the profiled model coefficients, the *inference performance predictor* first estimates the inference latency using our performance model designed in Sec. 3.1. It then guides our *GPU resource allocator* and *inference workload placer* to identify an *appropriate* GPU device with the *least* performance interference and the guaranteed SLOs from candidate GPUs for each inference workload. According to the cost-efficient GPU resource provisioning plan generated by Alg. 1, the *GPU device launcher* finally builds a GPU cluster and launches the Triton inference serving process for each DNN inference workload on the provisioned GPU devices. In addition, we configure the batch size of each workload by modifying the configuration file of Triton, and allocate the GPU resources to each Triton process using the `set_active_thread_percentage` command in MPS.

Dealing with performance prediction errors: Due to the prediction errors incurred by the *inference performance predictor*, our *GPU resource allocator* is likely to *over-provision* or *under-provision* GPU resources to DNN inference workloads. *iGniter* neglects the impact of resource over-provisioning and mainly deals with the resource under-provisioning as it can result in SLO violations. To solve such SLO violations, we leverage the technique of pre-launching a *shadow* Triton inference serving process for each workload on GPUs, which runs upon the remaining unallocated resources of a GPU device. The allocated GPU resources of the *shadow* inference process are actually set as the smaller value of the 10.0% of GPU resources (*i.e.*, the maximum prediction error measured in Sec. 5.2) and the remaining resources on a GPU device. In the beginning, the DNN inference requests are sent to the original Triton inference serving process. When the P99 latency of inference requests violates the latency SLOs, the upcoming requests are then *redirected* to the *shadow* inference process. In particular, the user clients continuously monitor the accumulated P99 latency with the allocated GPU resources every second and determine whether to switch the inference requests to the *shadow* inference process. We will validate the robustness of *iGniter* in handling the performance prediction errors of DNN inference workloads in Sec. 5.3.

TABLE 3: Configurations of three Apps with two types of SLOs, *i.e.*, latency (ms) and throughput (req/s) for four representative DNN inference workloads.

	SLOs	AlexNet	ResNet-50	VGG-19	SSD
App1	Latency	10	20	20	25
	Throughput	1200	400	300	150
App2	Latency	15	30	30	40
	Throughput	400	600	400	50
App3	Latency	20	40	40	55
	Throughput	800	200	200	300

5 Performance Evaluation

In this section, we evaluate *iGniter* by carrying out a set of prototype experiments with four representative DNN models (as listed in Table 3) on Amazon EC2 [23]. Our prototype experiments seek to answer the following questions:

- **Accuracy:** Can our inference performance model in *iGniter* accurately predict the performance of DNN inference workloads? (Sec. 5.2)
- **Effectiveness:** Can our GPU resource provisioning strategy in *iGniter* provide predictable DNN inference while saving the monetary cost in the cloud? (Sec. 5.3)
- **Overhead:** How much runtime overhead of workload profiling and algorithm computation does *iGniter* practically bring? (Sec. 5.4)

5.1 Experimental Setup

Configurations of GPU cluster: We set up a GPU cluster with 10 V100 GPU cards based on Triton using p3.2xlarge EC2 instances equipped with 1 NVIDIA V100 GPU. On each instance, we launch Triton inference serving process and its corresponding client with a constant request inter-arrival rate for each DNN inference workload. We measure the seven *hardware-specific* coefficients (*i.e.*, P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch}) using the `Nsight Systems` and `nvidia-smi` according to Sec. 3.1. The maximum power P, maximum frequency F, idle power p_{idle}, and available PCIe bandwidth B_{pcie} of V100 are 300 W, 1530 MHz, 53.5 W, and 10 GBps, respectively. The power coefficient α_f, scheduling coefficient α_{sch} and β_{sch} are profiled as -1.025, 0.00475 and -0.00902, respectively.

Configurations of DNN inference workloads: We select four representative DNN models as listed in Table 3. The AlexNet [24], ResNet-50 [25], and VGG-19 [26] models are used for image classification running on the ImageNet dataset [35], with a wide spectrum of inference latencies on the same GPU. The SSD [36] is used for object detection running on the VOC2012 dataset [37]. In total, we consider 12 DNN inference workloads with different latency SLOs and various throughput SLOs (*i.e.*, request arrival rates) for the three Apps (*i.e.*, App1, App2, and App3).

Baseline and metrics: We compare *iGniter* with the following three strategies: (1) **FFD$^+$:** the First-Fit Decreasing (FFD) algorithm with allocating the lower bound of GPU resources r^i_{lower}, which always allocates the lower bound of GPU resources r^i_{lower} and place the inference workloads

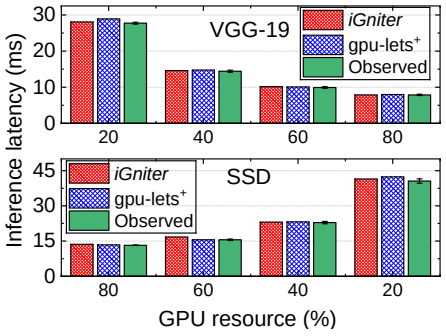

Fig. 11: Comparison of the observed and predicted inference latency of co-located VGG-19 and SSD with different allocated GPU resources and batch size set as 3 under the gpu-lets$^+$ and *iGniter* models.

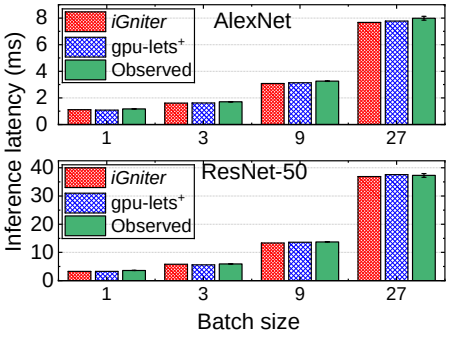

Fig. 12: Comparison of the observed and predicted inference latency of co-located AlexNet and ResNet-50 with 50% of allocated GPU resources and different batch sizes under the gpu-lets$^+$ and *iGniter* models.

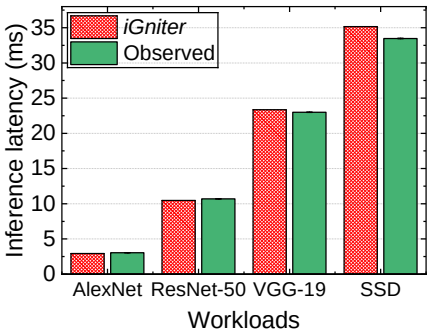

Fig. 13: Comparison of the observed and *iGniter* predicted inference latency of co-located AlexNet, ResNet-50, VGG-19 and SSD with 25% of allocated GPU resources and batch size set as 3.

using FFD; (2) **GSLICE$^+$:** the modified GSLICE [14] patched with our inference workload placement plan, which tunes the allocated GPU resources and batch sizes according to the online average latency and throughput for each inference workload on each GPU based on our workload placement plan; (3) **gpu-lets$^+$:** the modified spatial sharing strategy in gpu-lets [19], which allocates the amount of GPU resources maximizing the request throughput and places the inference workloads on the Best-Fit GPUs. As the large batch sizes cannot adapt to the low request arrival rate as elaborated in Sec. 2.3, we change the strategy of batch size configuration for GSLICE$^+$ and gpu-lets$^+$ to increase the batch size to *just* meet the request arrival rate. Moreover, we focus on two key metrics including the *monetary cost* and *SLO violations*. We define the P99 latency of an inference workload exceeding its latency SLO as a SLO violation.

5.2 Validating Inference Performance Model in *iGniter*

We evaluate the inference latency of AlexNet, ResNet-50, VGG-19 and SSD by varying the amount of GPU resources, batch size and the number of co-located inference workloads. We compare our *iGniter* performance model with the state-of-the-art gpu-lets$^+$ model [19]. We illustrate the observed inference latency with error bars of standard deviation by repeating the inference co-location experiments for three times.

Can *iGniter* accurately predict the inference latency with different amounts of GPU resources? We first examine the accuracy of predicted inference latency of VGG-19 and SSD by varying the allocated GPU resources with a fixed batch size (*i.e.,* 3) under gpu-lets$^+$ and *iGniter*. As shown in Fig. 11, *iGniter* can well predict the inference latency with a prediction error of 0.04% – 2.32% for VGG-19 and 0.89% – 7.61% for SSD compared with 1.30% – 4.19% and 0.02% – 4.43% under gpu-lets$^+$. Specifically, our predicted inference latency of SSD is basically higher than gpu-lets$^+$ and the observed latency. This is because the active time of SSD predicted by our model is higher than the actual active time, and the contention of the GPU power consumption and L2 cache utilization further makes it worse. However, gpu-lets$^+$ offline profiles the actual inference latency for all possible configurations when SSD is running alone which is more accurate than *iGniter*. In addition, the predicted inference latency of VGG-19 under gpu-lets$^+$ is basically higher than that under the *iGniter* and the observed latency. This is because gpu-lets$^+$ does not consider the contention of the GPU scheduler and power consumption, while the GPU frequency of VGG-19 drops from 1,530 MHz to 1,440 MHz due to GPU power contention, which makes the prediction error of gpu-lets$^+$ larger than *iGniter* for VGG-19.

Can *iGniter* accurately predict the inference latency with different batch sizes? We further examine the accuracy of predicted inference latency of AlexNet and ResNet-50 by varying the batch size with a fixed amount (*i.e.,* 50%) of GPU resources under gpu-lets$^+$ and *iGniter*. As depicted in Fig. 12, *iGniter* can basically predict the DNN inference latency with a prediction error of 3.91% – 5.90% for AlexNet and 1.10% – 9.29% for ResNet-50 compared with 2.67% – 6.23% and 0.78% – 9.76% of gpu-lets$^+$. In more detail, the predicted inference latency of AlexNet under *iGniter* is smaller than the observed latency. This is because the data loading and result feedback phases occupies a larger part (*i.e.,* 7% – 20%) in the inference latency for AlexNet than that for the other models (*i.e.,* 1% – 7%). It makes AlexNet share the PCIe bandwidth for a long period of time with other inference workloads, while we simply assume that the contention of the PCIe bandwidth can be negligible. Also, *iGniter* underestimates the inference latency of ResNet-50 with a prediction error of 9.29% when the batch size is set as 1. This is because the average kernel execution latency of ResNet-50 is relatively small (*i.e.,* 0.04 ms) which makes ResNet-50 being more sensitive to the contention of GPU scheduler than other workloads. As *iGniter* explicitly considers the contention of GPU scheduler, the average prediction error of *iGniter* (*i.e.,* 3.82%) can be smaller than that of gpu-lets$^+$ (*i.e.,* 4.15%) for ResNet-50.

Can *iGniter* adapt to the co-location of multiple (4+) inference workloads? As shown in Fig. 13, we observe that our *iGniter* model can accurately predict the inference latency of the four co-located workloads with a prediction error of 1.53% – 5.02%, while gpu-lets$^+$ cannot predict the inference latency of more than two inference workloads co-located on a GPU device. Specifically, our *iGniter* model is able to capture the inference on the GPU scheduler (Eq. (6)), L2 cache space (Eq. (8)), and power consumption (Eq. (9)) for multiple co-located inference workloads. Taking VGG-19 as an example, *iGniter* can well predict the inference latency with a prediction error of 4.19% when co-located

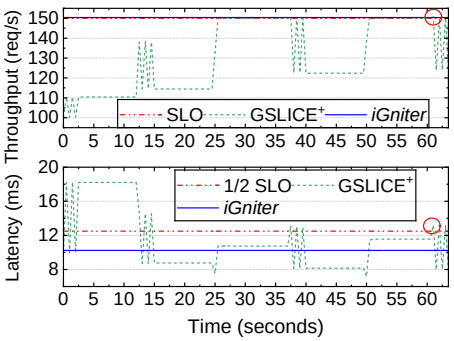

Fig. 14: Comparison of the inference latency and request throughput of SSD over time under the GSLICE$^+$ and *iGniter* strategies.

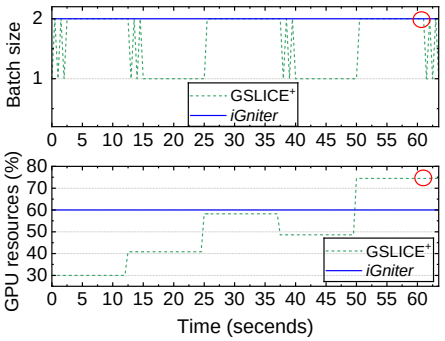

Fig. 15: Comparison of the allocated GPU resources and batch sizes for SSD over time under the GSLICE$^+$ and *iGniter* strategies.

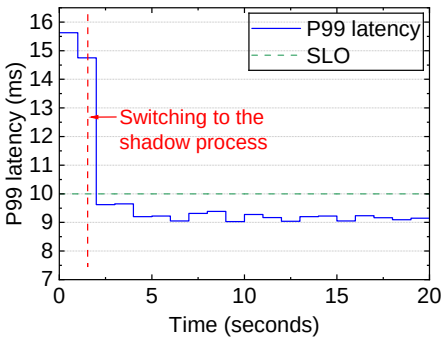

Fig. 16: The P99 inference latency of App1 of AlexNet over time as *iGniter* handles SLO violations.

TABLE 4: Comparison of the monetary cost and the number of P99 latency violations of inference workloads achieved by the gpu-lets$^+$, FFD$^+$, GSLICE$^+$, and *iGniter* resource provisioning strategies.

	gpu-lets$^+$	FFD$^+$	GSLICE$^+$	*iGniter*
Cost ($)	24.48	15.3	18.36	18.36
Violations	3	10	3	0

only with SSD (in Fig. 11) and 1.53% when co-located with three inference workloads (*i.e.*, AlexNet, ResNet-50, and SSD in Fig. 13), respectively. The rationale is that: *iGniter* can still predict the increase of GPU scheduling delay of VGG-19 from 0.19 ms to 0.36 ms, the decrease of GPU active time from 27.54 ms to 22.31 ms (allocated with 5% more GPU resources), and the drop of GPU frequency of VGG-19 from 1530 MHz to 1515 MHz, respectively, as co-located with two more workloads (*i.e.*, AlexNet, ResNet-50).

5.3 Effectiveness of GPU Resource Provisioning Strategy in *iGniter*

To illustrate the effectiveness of *iGniter*, we conduct extensive experiments with the inference workloads in Table 3. Specifically, we leverage *iGniter* to obtain the resource provisioning plans of these 12 inference workloads, and launch multiple EC2 instances to execute the workloads and then measure the inference performance. We measure the P99 latency of inference workloads within 30 seconds. As GSLICE$^+$ is an online adjustment solution, we always take the resource provisioning plan after *five* adjustments of GPU resources. Similarly, *iGniter* preserves the resource provisioning plan *after* dealing with prediction error. As shown in Table 4, *iGniter* guarantees that the P99 inference latency of all inference workloads can meet the latency SLOs and while saving up to 25% of monetary cost as compared with gpu-lets$^+$.

How can *iGniter* guarantee the latency SLOs? As shown in Table 4, *iGniter* can guarantee the latency SLOs of all inference workloads while other strategies fail to guarantee SLO for 3 – 10 inference workloads. FFD$^+$ does not consider the performance interference of co-located workloads, which leads to the largest number of SLO violations. Compared with FFD$^+$, *iGniter* provisions an additional 25% of GPU resources to *proactively* eliminate SLO violations caused by the performance interference. Though gpu-lets$^+$ provisions the largest amount of GPU resources, there are still three inference workloads violating their latency SLOs. This is because the performance interference of inference workloads can negatively affect both the inference latency and throughput. While gpu-lets$^+$ simply assigns the request arrival rate based on its profiled throughput as the inference workload running alone, *iGniter* considers the impacts of performance interference on both of them.

As an online adjustment solution, GSLICE$^+$ still causes 3 violations even though using our workload placement plan. Fig. 14 and Fig. 15 depict the performance and resource allocations of App1 of SSD co-located with App3 of VGG-19 over time. We observe that GSLICE$^+$ can meet the performance SLOs during 25.5 – 37.5 seconds and 50.5 – 61 seconds, but it quickly violates the performance SLOs after these time periods. This is because the *interference-unaware* strategy (*i.e.*, GSLICE$^+$) *separately* tunes the allocated GPU resources and batch size according to the online measured performance and the tuning threshold. Specifically, the average inference latency (*i.e.*, 10.7 ms) is lower than the $\frac{1}{2}$SLO (*i.e.*, 12.5 ms) which exceeds the threshold (*i.e.*, 10%) during 25.5 – 37.5 seconds, and thus it triggers GSLICE$^+$ to reduce the allocated GPU resources. Also, GSLICE$^+$ adjusts the GPU resources of VGG-19 to 100% at the 51-th second without considering SSD, and the resources are successfully allocated at the 61-th second (the red circle in Fig. 14 and Fig. 15). Later, the performance of SSD starts to *oscillate frequently* as the throughput cannot meet the SLO with batch size set as 1 and the average latency cannot meet the $\frac{1}{2}$SLO with batch size set as 2. In contrast, *iGniter* leverages our analytical inference performance model to *proactively* provisions the adequate amount of GPU resources and configure an appropriate batch size at the beginning for DNN inference workloads.

Can *iGniter* deal with the performance prediction errors? The prediction error handling mechanism in *iGniter* further guarantees the latency SLOs. In our experiments, *iGniter* only tunes the allocated GPU resources for two inference workloads. Fig. 16 depicts the P99 latency of App1 of AlexNet over time which is co-located with App2 of ResNet-50 and App2 of SSD. In more detail, the P99 latency of AlexNet at the first second is 15.6 ms which is higher than the SLO (*i.e.*, 10 ms). In the next 0.5 second, *iGniter* collects request latency data and judges whether it violates the latency SLO. If an SLO violation occurs, *iGniter* switches the violated inference workload to the shadow Triton process at the 1.5-th second. After another 1.5 seconds, the P99

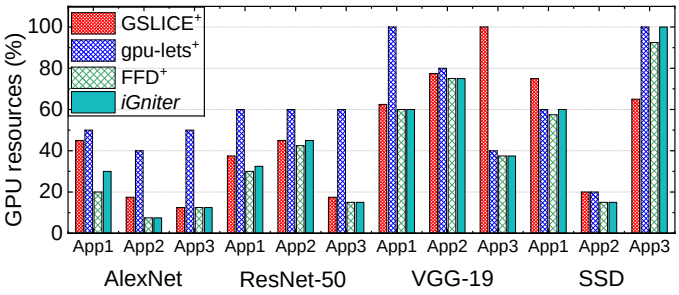

Fig. 17: Comparison of allocated GPU resources achieved by the gpu-lets$^+$, FFD$^+$, GSLICE$^+$, and *iGniter* strategies.

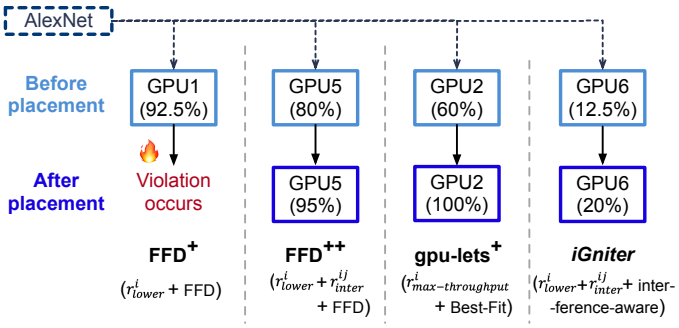

Fig. 18: Comparison of the inference workload (*i.e.*, App2 of AlexNet) placement decisions achieved by the FFD$^+$, gpu-lets$^+$, FFD^{++}, and *iGniter* resource provisioning strategies.

latency of App1 of AlexNet can be finally guaranteed. As we have pre-launched the *shadow* Triton process as elaborated in Sec. 4.2, *iGniter* does not require spending an extra 10 seconds in launching a new Triton process as in GSLICE$^+$.

How can *iGniter* reduce the monetary cost? We compare the allocated GPU resources of *iGniter* with that of the other three strategies (*i.e.*, GSLICE$^+$, FFD$^+$, and gpu-lets$^+$) in Fig. 17. We observe that the GPU resources allocated by gpu-lets$^+$ for each inference workload are larger or equal to *iGniter*. This is mainly due to the following facts: *First*, gpu-lets$^+$ allocates the amount of GPU resources to maximize the throughput, while *iGniter* allocates the amount of GPU resources at the appropriate batch size to *just meet* the performance SLOs. For example, gpu-lets$^+$ provisions 60% of GPU resources (*i.e.*, the most-efficient amount of GPU resources) for App1 of ResNet-50 and then sets the batch size as 2. In contrast, *iGniter* sets the appropriate batch size as 4 according to the latency SLO and request arrival rate and then provisions 32.5% of GPU resources to meet its latency SLO. *Second*, gpu-lets$^+$ only allows two inference workloads to be co-located, while *iGniter* allows multiple (more than 2) inference workloads concurrently executed on a GPU device. Third, gpu-lets$^+$ allows only *five* choices (*i.e.*, 20%, 40%, 50%, 60%, 80%) of GPU resources allocated to the inference workloads, while *iGniter* allocates the inference workloads with an amount of GPU resources by selecting the number of GPU allcation units (*i.e.*, 2.5%). For example, gpu-lets$^+$ and *iGniter* provision the App3 of VGG-19 with 40% and 37.5% of GPU resources, respectively.

Though GSLICE$^+$ uses our workload placement plan, it provisions more or equal amounts of GPU resources than *iGniter* for all inference workloads except App3 of SSD which violates its latency SLO. This is because GSLICE$^+$ does not reduce its allocated GPU resources, as long as an inference workload meets its latency SLOs and the tuning threshold. FFD$^+$ provisions less or equal amounts of GPU resources than *iGniter* as it always allocates the lower bound (r^i_{lower}) of GPU resources to inference workloads.

The *inference workload placer* elaborated in Sec. 4.2 in *iGniter* further saves the amount of allocated GPU resources. Fig. 18 illustrates the placement decisions of App2 of AlexNet made by FFD$^+$, FFD^{++} (FFD$^+$ using `alloc_gpus`, Alg. 2), gpu-lets$^+$, and *iGniter*. Specifically, FFD$^+$ places App2 of AlexNet onto the GPU1 according to the lower bound of GPU resources (*i.e.*, r^i_{lower}) which inevitably causes SLO violations due to the overlooked performance interference. FFD^{++} places such an AlexNet workload onto the GPU5 with 15% of GPU resources according to the first-fit GPU that still has an amount (*i.e.*, $r^i_{lower} + r^{ij}_{inter}$ which is calculated by our `alloc_gpus` algorithm) of GPU resources. As the most-efficient amount of GPU resources (*i.e.*, $r^i_{max\_throughput}$) for App2 of AlexNet is 40%, gpu-lets$^+$ places such an AlexNet workload onto the GPU2 which is selected as the best-fit GPU device. In general, gpu-lets$^+$ allocates more GPU resources than the other strategies as it mainly focuses on improving the inference throughput. In contrast, *iGniter* places the AlexNet workload onto the GPU6 with the least amount of GPU resources (7.5%) while guaranteeing the latency SLOs. This is because *iGniter* greedily places the inference workload onto the GPU with the least performance interference and allocates the GPU resources that *just meet* the performance SLOs.

5.4 Runtime Overhead of *iGniter*

We evaluate the runtime overhead of *iGniter* in terms of the profiling overhead of DNN inference workloads and the computation time of *iGniter* resource provisioning strategy (*i.e.*, Alg. 1). Specifically, we launch a p3.2xlarge EC2 instance equipped with 1 NVIDIA V100 GPU to profile workload-specific coefficients for each inference workload. We also profile the hardware-specific coefficients with VGG-19 *only once for a given GPU type* and the profiling time is only 229 seconds. With 11 configurations for each DNN model, we profile all 8 workload-specific coefficients for each inference workload. The profiling time of AlexNet [24], ResNet-50 [25], VGG-19 [26], and SSD [36] models are 231, 247, 240, and 237 seconds, respectively. The experiment results above show that the profiling overhead are within several (around 4) minutes. Each inference model requires *profiling only once* to obtain the essential model coefficients. The workload profiling overhead is far less than gpu-lets [19] which spends over several hours in our experiments. After obtaining the performance model coefficients, we also run our *iGniter* strategy in Alg. 1 on the p3.2xlarge EC2 instance. The computation overhead of *iGniter* for 12 workloads in Table 3 is negligible (*i.e.*, merely 3.69 milliseconds). This is because the computation time of Alg. 1 is quadratic to the number of DNN inference workloads, as analyzed in Sec. 4.1. As a result, the runtime overhead of our *iGniter* strategy can be acceptable in practice.

6 Related Work

Achieving predictable DNN inference on GPUs: There have been a number of works on guaranteeing DNN in-

ference latency SLOs on GPUs. In the scenario of ***disabling GPU sharing*** (*i.e.,* a GPU serves one DNN inference at a time), Clipper [17] proposes caching, adaptive batch size, and dynamic model selection techniques to achieve low-latency and high-throughput DNN inference. To improve the resource efficiency, Swayam [38] designs a resource auto-scaling protocol using a *profiled* resource estimation model. In the scenario of ***temporal sharing*** of GPUs, Nexus [10] proposes batching-aware scheduling based on Clipper [17] to place user requests on GPUs, with the aim of improving the GPU utilization. Clockwork [13] further designs fine-grained request-level scheduling to order user requests based on their *tight* latency SLOs. While *iGniter* shares the adaptive batching techniques with the prior works above, the objective of *iGniter* is cost-efficiently guaranteeing the performance (*i.e.,* latency and throughput) SLOs based on GPU *spatial sharing*, instead of maximizing the request throughput of inference workloads.

To further reduce the monetary cost of DNN inference, two more recent works (*i.e.,* Cocktail [12], INFaaS [18]) design the heterogeneous instance/accelerator selection, resource autoscaling, and dynamic model-variants selection techniques for cost-effective resource provisioning. These techniques above can be incorporated into *iGniter* to further save the inference budget. In addition, our *SM-level* resource scaling in *iGniter* to handle the performance prediction errors is more *fine-grained* than the *device-level* resource scaling in INFaaS and Cocktail.

In the scenario of ***spatial sharing*** of GPUs that can fully utilize the GPU resources, Scrooge [11] combines the CUDA streams and batching techniques to efficiently pack the inference workloads into VMs for ensuring the performance SLOs of DL-based media applications. However, it requires profiling performance (*i.e.,* latency and throughput) with *all possible* (*e.g.,* over hundreds) configurations of batch sizes and concurrency levels which inevitably brings heavy profiling overhead. Based on the latest multi-instance GPU (MIG) [39] featured A100 GPUs, MIG-serving [40] proposes an algorithm pipeline that produces a set of GPU partitions and DNN inference deployments to most efficiently meet SLOs. In contrast, *iGniter* leverages the NVIDIA MPS to spatially share the GPUs and builds an analytical inference performance model in terms of the batch size and allocated GPU resources. In particular, our model requires only workload profiling with a limited number (*i.e.,* 11) of configurations which is *lightweight* compared with Scrooge [11].

To maximize the request throughput while guaranteeing the latency SLOs, two more recent works (*i.e.,* GSLICE [14], gpu-lets [19]) *separately* adjust the allocated GPU resources and batch size for each inference workload. Nevertheless, GSLICE [14] is oblivious to performance interference and thus it tends to cause uncontrollable long-tail latency due to the shared resource contention and GPU resource under-provisioning. Though gpu-lets [19] explicitly takes into account the performance interference of inference workloads, it only considers the interference impacts to the newly-placed workload while ignoring the impacts to the workloads that have already been placed on GPUs. *iGniter* differs from the two works above in that: *iGniter jointly* optimizes the GPU resource allocation and batch size configuration by proactively considering (minimizing) the performance interference among co-located inference workloads, with the aim of achieving predictable DNN inference while reducing the inference budget.

Modeling performance interference in clouds: There have been prior works on modeling the performance interference [41] and hardware heterogeneity [42] among cloud VMs based on the CPU architecture. Several recent studies have been devoted to modeling the performance interference among co-located VMs particularly based on ***temporal sharing*** of GPUs. For instance, Xu *et al.* [43] build a random forest regression model with a set of factors such as GPU/memory utilization and the average kernel length. As DNN training and inference workloads become prevailing in cloud datacenters [44], Horus [45] leverages GPU utilization to estimate the performance interference among co-located *DNN training* jobs through fitting a quadratic function, while *iGniter* focuses on modeling the performance of *DNN inference* workloads using a set of easily-accessible system and workload metrics.

Different from the interference above caused by the context switching in the *preemptible temporal sharing* of GPUs, NVIDIA MPS allows workloads to ***spatially share*** the resources of a GPU device, which leads to the contention of shared GPU resources. Prophet [46] designs a performance interference model to characterize the contention of GPU resources and DRAM bandwidth as well as PCIe bandwidth in the *default* mode of MPS [47]. Based on the *MPS with limited GPU resources*, gpu-lets [19] builds a linear regression model in terms of the L2 cache utilization and the DRAM bandwidth utilization to predict the latency increases for only *two* inference workloads. However, it requires profiling a number (which is likely to be thousands) of possible workload configurations, which brings heavy profiling overhead. Different from the prior works above, *iGniter* builds an analytical model to predict the performance interference among multiple (*i.e.,* more than 2) DNN inference workloads through a *lightweight* workload profiling (as discussed in Sec. 3.1). Moreover, our *iGniter* model *comprehensively* considers the severe contention of the GPU scheduler, the L2 GPU cache space, and the GPU power consumption among co-located inference workloads.

7 Conclusion and Future Work

In this paper, we present the design and implementation of *iGniter*, an interference-aware GPU resource provisioning framework for achieving predictable performance of DNN inference while reducing the monetary cost in the cloud. Specifically, we build a lightweight analytical performance model by identifying the key system and workload metrics that can impact the performance interference of DNN inference workloads co-located on GPUs. Based on such a performance model, we further design a cost-efficient GPU resource provisioning strategy, which jointly optimizes the GPU resource allocation and batch size configuration for DNN inference workloads to greedily minimize the performance interference. Extensive prototype experiments on Amazon EC2 demonstrate that *iGniter* is able to guarantee the performance SLOs for cloud-based DNN inference workloads, while cutting down the monetary cost by up to

25% compared with the state-of-the-art resource provisioning strategies.

We plan to extend *iGniter* in the following two directions: (1) supporting *iGniter* with multiple types of GPU hardwares, and (2) provisioning multiple GPU devices for a DNN inference workload with an extremely large request arrival rate.

REFERENCES

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] 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," in *Proc. of NeurIPS*, Dec. 2018, pp. 1–8.

[3] NVIDIA. (2019, May) Intel Inference NVIDIA GPUs. [Online]. Available: https://blogs.nvidia.com/blog/2019/05/21/intel-inference-nvidia-gpus/

[4] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep Interest Evolution Network for Click-Through Rate Prediction," in *Proc. of AAAI*, vol. 33, no. 01, 2019, pp. 5941–5948.

[5] NVIDIA. (2018, May) JD AI Video Inferencing. [Online]. Available: https://blogs.nvidia.com/blog/2018/02/13/jd-ai-video-inferencing/

[6] Amazon. (2021, Nov.) Amazon SageMaker. [Online]. Available: https://aws.amazon.com/sagemaker/

[7] Google Cloud. (2021, Nov.) Vertex AI. [Online]. Available: https://cloud.google.com/vertex-ai

[8] Omdia. (2021, Aug.) NVIDIA Maintains Dominant Position In 2020 Market. [Online]. Available: https://omdia.tech.informa.com/pr/2021-aug/nvidia-maintains-dominant-position-in-2020-market-for-ai-processors-for-cloud-and-data-center

[9] A. Priestley. (2020, Oct.) Forecast Analysis: Discrete GPUs, Worldwide. [Online]. Available: https://www.gartner.com/en/documents/3991464/forecast-analysis-discrete-gpus-worldwide

[10] 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 *Proc. of ACM SOSP*, Oct. 2019, pp. 322–337.

[11] Y. Hu, R. Ghosh, and R. Govindan, "Scrooge: A Cost-Effective Deep Learning Inference System," in *Proc. of ACM SOCC*, Nov. 2021, pp. 624–638.

[12] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, M. T. Kandemir, and C. R. Das, "Cocktail: A Multidimensional Optimization for Model Serving in Cloud," in *Proc. of USENIX NSDI*, Apr. 2022, pp. 1–17.

[13] 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 *Proc. of USENIX OSDI*, Nov. 2020, pp. 443–462.

[14] A. Dhakal, S. G. Kulkarni, and K. K. Ramakrishnan, "GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform," in *Proc. of ACM SOCC*, Oct. 2020, pp. 492–506.

[15] NVIDIA. (2021, Jun.) NVIDIA Multi-Process Service. [Online]. Available: https://docs.nvidia.com/deploy/mps

[16] W. Zhang, Q. Chen, N. Zheng, W. Cui, K. Fu, and M. Guo, "Towards QoS-awareness and Improved Utilization of Spatial Multitasking GPUs," *IEEE Transactions on Computers*, pp. 1–14, 2021.

[17] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency Online Prediction Serving System," in *Proc. of USENIX NSDI*, Mar. 2017, pp. 613–627.

[18] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated Model-less Inference Serving," in *Proc. of USENIX ATC*, Jul. 2021, pp. 397–411.

[19] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Multi-model Machine Learning Inference Serving with GPU Spatial Partitioning," *arXiv preprint arXiv:2109.01611*, 2021.

[20] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2014.

[21] NVIDIA. (2021, Nov.) NVIDIA Triton Inference Server. [Online]. Available: https://github.com/triton-inference-server/server

[22] S. Kim, S. Oh, and Y. Yi, "Minimizing GPU Kernel Launch Overhead in Deep Learning Inference on Mobile GPUs," in *Proc. of HotMobile*, Feb. 2021, pp. 57–63.

[23] Amazon. (2021, Nov.) Amazon Elastic Compute Cloud (Amazon EC2). [Online]. Available: https://aws.amazon.com/ec2/

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of IEEE CVPR*, Jun. 2016, pp. 770–778.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. of ICLR*, May 2015, pp. 1–14.

[27] NVIDIA. (2021, Nov.) NVIDIA TensorRT. [Online]. Available: https://developer.nvidia.com/tensorrt

[28] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs," in *Proc. of IEEE RTAS*, Jul. 2019, pp. 29–41.

[29] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU," in *Proc. of ICPP*, Oct. 2013, pp. 826–833.

[30] NVIDIA. (2021, Nov.) NVIDIA Nsight Systems. [Online]. Available: https://developer.nvidia.com/nsight-systems

[31] ——. (2019, May) NVIDIA System Management Interface. [Online]. Available: https://blogs.nvidia.com/blog/2019/05/21/intel-inference-nvidia-gpus/

[32] Wikipedia. (2021, Nov.) Regression Analysis. [Online]. Available: https://en.wikipedia.org/wiki/Regression_analysis

[33] NVIDIA. (2021, Nov.) NVIDIA Nsight Compute. [Online]. Available: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

[34] D. S. Johnson, "Near-Optimal Bin Packing Algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1973.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. of IEEE CVPR*, Jun. 2009, pp. 248–255.

[36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *Proc. of ECCV*, Oct. 2016, pp. 21–37.

[37] M. Everingham and J. Winn, "The Pascal Visual Object Classes Challenge 2012 (VOC2012) Development Kit," Tech. Rep., May 2012.

[38] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services with Resource Efficiency," in *Proceedings of Middleware*, Dec. 2017, pp. 109–120.

[39] NVIDIA. (2021, Jun.) NVIDIA Multi-Instance GPU User Guide. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

[40] C. Tan, Z. Li, J. Zhang, Y. Cao, S. Qi, Z. Liu, Y. Zhu, and C. Guo, "Serving DNN Models with Multi-Instance GPUs: A Case of the Reconfigurable Machine Scheduling Problem," *arXiv preprint arXiv:2109.11067*, 2021.

[41] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 3012–3025, 2014.

[42] F. Xu, F. Liu, and H. Jin, "Heterogeneity and Interference-Aware Virtual Machine Provisioning for Predictable Performance in the Cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2470–2483, 2016.

[43] X. Xu, N. Zhang, M. Cui, M. He, and R. Surana, "Characterization and Prediction of Performance Interference on Mediated Pass through GPUs for Interference-Aware Scheduler," in *Proc. of USENIX HotCloud*, Jul. 2019.

[44] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training," in *Proc. of ICPP*, Aug. 2019, pp. 1–11.

[45] 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.

[46] 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 *Proc. of ACM ASPLOS*, Apr. 2017, pp. 17–32.

[47] 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.

Fei Xu received the B.S., M.E., and Ph.D. degrees in 2007, 2009, and 2014, respectively, all from the Huazhong University of Science and Technology (HUST), Wuhan, China. He received Outstanding Doctoral Dissertation Award in Hubei province, China, and ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award in 2015. He is currently an associate professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests include cloud computing and datacenter, virtualization technology, and distributed systems.

Jianian Xu received his B.S. degree in Polymer Materials and Engineering from Qingdao University of Science and Technology in 2019. He is currently working toward the master's degree in the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests focus on cloud computing and distributed machine learning systems.

Jiabin Chen received his B.S. degree in Optoelectronic Information Science and Engineering from Harbin Institute of Technology, Weihai in 2019. He is currently working toward the master's degree in the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests focus on cloud computing and distributed machine learning systems.

Li Chen received the BEngr degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2012 and the MASc degree from the Department of Electrical and Computer Engineering, University of Toronto, in 2014 and the PhD degree in computer science and engineering from the Department of Electrical and Computer Engineering, University of Toronto, in 2018. She is currently an assistant professor with the Department of Computer Science, School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, USA. Her research interests include big data analytics systems, cloud computing, datacenter networking, and resource allocation.

Ruitao Shang received her B.S. degree in Computer Science from East China Normal University (ECNU) in 2020. She is currently pursuing her MS degree in Computer Science in the School of Computer Science and Technology at ECNU. Her current research interests focus on cloud computing and distributed machine learning systems.

Zhi Zhou received the B.S., M.E., and Ph.D. degrees in 2012, 2014, and 2017, respectively, all from the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the School of Computer Science and Engineering at Sun Yat-sen University, Guangzhou, China. In 2016, he was a visiting scholar at University of Göttingen. He was nominated for the 2019 CCF Outstanding Doctoral Dissertation Award, the sole recipient of the 2018 ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award, and a recipient of the Best Paper Award of IEEE UIC 2018. His research interests include edge computing, cloud computing, and distributed systems.

Fangming Liu (S'08, M'11, SM'16) received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, the First Class Prize of Natural Science of Ministry of Education in China, as well as the Second Class Prize of National Natural Science Award in China.

APPENDIX

Proof. By substituting Eq. (1) into Eq. (14), we have $\frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback} \geq \sum_{j\in\mathcal{J}} t^{ij}_{gpu}$. The inference latency increases as the allocated GPU resources decrease with a given batch size. To minimize the amount of GPU resources, we set the GPU latency to the maximum value as

$$\sum_{j\in\mathcal{J}} t^{ij}_{gpu} = \frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback}. \tag{19}$$

By substituting Eq. (19), Eq. (2), and Eq. (3) into Eq. (13), we have $b^i \geq \frac{T^i_{slo}\cdot R^i\cdot B_{pcie}}{2\cdot(B_{pcie}+\cdot R^i\cdot d^i_{load})}$. As a larger batch size results in a higher inference latency given an amount of allocated GPU resources, we simply choose the *appropriate* batch size b^i_{appr} that *just meets* the arrival rate, which is given by

$$b^i_{appr} = \left\lceil \frac{T^i_{slo}\cdot R^i\cdot B_{pcie}}{2\cdot(B_{pcie}+R^i\cdot d^i_{load})} \right\rceil.$$

By substituting b^i_{appr}, Eq. (1), Eq. (4), Eq. (5), Eq. (6), and Eq. (11) into Eq. (14), we have

$$\begin{aligned} r &\geq \frac{k^i_1\cdot(b^i_{appr})^2 + k^i_2\cdot b^i_{appr} + k^i_3}{\left(\frac{T^i_{slo}}{2} - \frac{(d^i_{load}+d^i_{feedback})\cdot b^i_{appr}}{B_{pcie}}\right)\cdot\frac{f}{F} - k^i_5 - k^i_{sch}} - k^i_4 \\ &\geq \frac{k^i_1\cdot(b^i_{appr})^2 + k^i_2\cdot b^i_{appr} + k^i_3}{\frac{T^i_{slo}}{2} - \frac{(d^i_{load}+d^i_{feedback})\cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch}} - k^i_4. \end{aligned}$$

As the GPU resources are allocated in units of r_{unit} which is 2.5% for NVIDIA V100 GPUs, the lower bound r_{lower} of GPU execution resource can be calculated by

$$r^i_{lower} = \left\lceil \frac{\gamma^i}{\delta^i\cdot r_{unit}} - \frac{k^i_4}{r_{unit}} \right\rceil \cdot r_{unit},$$

where $\gamma^i = k^i_1\cdot(b^i_{appr})^2 + k^i_2\cdot b^i_{appr} + k^i_3$ and $\delta^i = \frac{T^i_{slo}}{2} - \frac{(d^i_{load}+d^i_{feedback})\cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch}$. □