PERSEUS: Characterizing Performance and Cost of Multi-Tenant Serving for CNN Models

Matthew LeMay  
Worcester Polytechnic Institute  
mlemay@wpi.edu

Shijian Li  
Worcester Polytechnic Institute  
sli8@wpi.edu

Tian Guo  
Worcester Polytechnic Institute  
tian@wpi.edu

Abstract—Deep learning models are increasingly used for end-user applications, supporting both novel features such as facial recognition, and traditional features, e.g. web search. To accommodate high inference throughput, it is common to host a single pre-trained Convolutional Neural Network (CNN) in dedicated cloud-based servers with hardware accelerators such as Graphics Processing Units (GPUs). However, GPUs can be orders of magnitude more expensive than traditional Central Processing Unit (CPU) servers. These resources could also be under-utilized facing dynamic workloads, which may result in inflated serving costs. One potential way to alleviate this problem is by allowing hosted models to share the underlying resources, which we refer to as multi-tenant inference serving. One of the key challenges is maximizing the resource efficiency for multi-tenant serving given hardware with diverse characteristics, models with unique response time Service Level Agreement (SLA), and dynamic inference workloads. In this paper, we present PERSEUS, a measurement framework that provides the basis for understanding the performance and cost trade-offs of multi-tenant model serving. We implemented PERSEUS in Python atop a popular cloud inference server called Nvidia TensorRT Inference Server. Leveraging PERSEUS, we evaluated the inference throughput and cost for serving various models and demonstrated that multi-tenant model serving led to up to 12% cost reduction.

Keywords—DNN inference, multi-tenancy, performance

I. INTRODUCTION

Infrastructure-as-a-Service (IaaS), has emerged as a popular option for training and deploying deep learning models, due to their pay-as-you-go pricing models and wide selection of hardware. The increasing usage of Convolutional Neural Network (CNN) models in computer vision applications requires efficient utilization of cloud resources. Consequently, understanding the cost and performance trade-offs of serving CNN model inference requests with various cloud hardware has garnered interest from researchers [1, 2].

However, the typical method of serving a CNN model with dedicated resources may lead to underutilized resources, especially when inference workloads vary. Such inefficiency often leads to higher monetary costs; the problem becomes more prominent when inference serving systems use expensive hardware accelerators such as Graphics Processing Units (GPUs) for higher throughput. One potential way to improve resource efficiency is supporting multi-tenant inference serving, in which models with different resource requirements share the underlying hardware.

As such, it is also possible to decrease serving costs by multiplexing CNN models on previously underutilized servers.

In this paper, we first show that multi-tenant model serving can achieve higher resource utilization and lead to promising cost savings, without violating performance guarantees for serving CNN models. Leveraging our measurement infrastructure called PERSEUS, we quantified the end-user perceived latency and throughput, as well as serving cost of running two representative CNN models on Google Cloud Platform’s Compute Engine. PERSEUS highlights the impacts on performance associated with multi-tenant model serving and examines the performance and cost tradeoffs of inference serving with different hardware configurations.

Previous literature explored the potential of using Functions-as-a-Service to achieve better resource utilization and scalability [1, 3–8]. Other works have explored the use of predictive scaling [9], Quality of Service (QoS) aware scheduling [10–12], GPU primitive sharing [13], and edge-based techniques [14–17] to improve serving efficiency. Our work complements prior research by providing the basis for understanding the performance implications and for improving resource utilization of cloud-based inference servings. We make the following key contributions.

• Our study demonstrates the need for multi-tenant model serving, and shows up to 12% cost savings when appropriately mixing inference workloads.

• We designed and implemented a suite of tools, collectively referred to as PERSEUS, that facilitates further evaluation of performance and cost trade-offs for new model serving scenarios, such as running new CNN models on different GPUs.

• We identify a number of aspects, including inefficient framework supports for CPU inference and for model caching, that hinder the observed inference performance. Our findings shed light on and pave the way for complementary research such as resource provisioning and load balancing for model serving.

1https://github.com/cake-lab/perseus
The remainder of this paper is structured as follows: Section II introduces the key concepts underpinning CNN model serving systems and discusses related work. Section III presents the problem statement followed by the design of PERSEUS and our measurement methodology for characterizing multi-tenant model serving, as presented in Section IV. Finally, we summarize the findings of our research and potential directions in Section V.

II. BACKGROUND AND RELATED WORK

There are numerous existing frameworks [1, 8, 18–23] and services [24–26] for supporting inference serving in cloud environments. We briefly describe these inference systems and common deployment practices. Then we discuss the hardware in which inference serving platforms leverage and holistic techniques for evaluating inference serving systems.

A. Inference Serving Frameworks

Inference serving frameworks have evolved to support a wide array of use cases, libraries, and platforms. TensorFlow Serving [20] is one of the initial open-source inference serving systems that leverages GPUs. TensorFlow Serving also supports multi-model deployments and provides an endpoint for prediction, but requires models to be trained using TensorFlow explicitly. Other frameworks such as PredictionIO and RedisAI [22, 23] allow the serving of models trained using different frameworks. Further, frameworks such as Nvidia’s TensorRT Inference Server [21] provide hardware-specific inference optimizations, e.g., for Nvidia’s GPUs.

Several frameworks have evolved to incorporate additional features aiming to improve performance of inference serving. Clipper [18] adds additional functionality to ensure SLAs and to achieve better prediction accuracy. MArk [1] and Barista [8] leverage Functions-as-a-Service (FaaS) to handle and scale transient workloads in order to maintain SLAs. INFaaS [19] shares models and hardware across applications by optimizing model deployment and autoscaling mechanisms. However, INFaaS focuses on a single VM configuration of either CPU and GPU, and uses GPU memory constraints to scale each model serving independently. This can lead to resource under-utilization especially when models serve dynamic inference requests. In contrast, we explore the inference cost savings of sharing resources without constraints through evaluating resource footprints of different model-hardware configurations.

| GPU Type       | Memory (GB) | Memory Bandwidth | Cuda Cores |
|----------------|-------------|------------------|------------|
| Nvidia Tesla P4 | 8           | 192 GB/s         | 2560       |
| Nvidia Tesla V4 | 16          | 320 GB/s         | 2560       |

Table I: Overview of evaluated Nvidia GPU devices.

| Machine Type | GPU Type       | GPU Count | Cost ($/hour) |
|--------------|----------------|-----------|---------------|
| n1-standard-8 | Nvidia Tesla P4 | 1         | 0.688         |
| n1-standard-8 | Nvidia Tesla P4 | 2         | 1.108         |
| n1-standard-8 | Nvidia Tesla T4 | 1         | 0.933         |
| n1-standard-8 | Nvidia Tesla T4 | 2         | 1.598         |
| n1-standard-8 | N/A             | N/A       | 0.268         |

Table II: Cloud server unit cost with GPU configurations from Google Cloud us-central1-a, as of November 26th, 2019.

B. Inference Serving Hardware

The abundance of commodity CPU servers in the cloud makes them ideal candidates for serving inference requests [27–29], while the emergence of hardware accelerators provide new opportunities and challenges. Among the plethora of accelerators, GPUs have become the most popular type and are closely associated with deep learning. Manufacturers have been making highly specialized GPUs for different deep learning tasks, such as Nvidia P4 GPU for inference jobs. Table I shows hardware specifications of two such GPUs. In this paper, we chose to focus on GPU inference for three reasons. First, GPUs are widely used in deep learning, particularly in the cloud environment. Second, GPUs exhibit intricate advantages and shortcomings compared to CPUs. For example, GPUs are ideal for highly parallel computation such as matrix multiplication which dominates CNN inference, while their performance are fundamentally constrained by limited GPU memory and slower memory transfer between CPU and GPU. Third, cloud-based GPUs are much more expensive, leading to large room for improvements of monetary cost.

C. Inference Serving Deployments

Deploying a CNN model to a pre-provisioned server requires developers to adhere to a given framework’s workflow. Namely, pre-trained models, with their weights and labels, must be exported into a format supported by the serving framework. Inference servers commonly expose endpoints such as REST, gRPC, or client API interface, which can be used to query a model [1, 7, 8, 18–26]. In systems that support autoscaling [1, 7, 8, 18, 19], middleware manages provisioning and acts as a single endpoint which routes requests to individual model serving.

Several major cloud providers offer managed inference serving frameworks such as Amazon’s SageMaker, for deploying a single model in an isolated environment [24–26]. These services abstract the deployment process and provide high-level tools for autoscaling individual models. Amazon’s Elastic Inference introduced the ability to acquire and attach a portion of a GPU’s resource to a SageMaker instance [30], further reducing over-provisioning.
Performing inference on chosen cloud server configurations. The first-order goal of inference serving is latency. Adhering to latency SLAs is one of the key challenges of inference serving, especially for applications that require real-time performance. Consequently, latency determines the viability of performing inference with a given configuration. SLA compliance is commonly measured by verifying that the 95th or 99th percentile of the end-to-end response time of recent requests is below a predefined threshold [1, 9].

There are a plethora of choices when deploying inference serving systems; therefore, it is important to determine a model’s characteristics for a given framework in a specific system. The first-order goal of inference serving is latency. Adhering to latency SLAs is one of the key challenges of inference serving, especially for applications that require real-time performance. Consequently, latency determines the viability of performing inference with a given configuration.

The monitoring stack was composed of Prometheus version 1.2.10 hosting Nvidia’s TensorRT Inference Server version 1.6.0, cAdvisor version 0.33.0 and Node exporter version 0.18.1 were used to collect the server’s resource data from a given directory, stores the preprocessed data in the client’s memory, and then generates and sends inference requests to the inference server. The number of clients in conjunction with its computation capacity determines the peak throughput ($\lambda$) in inference requests per second for a given model. The client is able to achieve an accurate estimation without using server-side statistics. The inference server is based on Nvidia’s TensorRT Inference Server [21] and includes two additional components cAdvisor and Node exporter that aggregate and export the performance characteristics and resource utilization data such as GPU, CPU and network utilization of running containers. We further use Prometheus in conjunction with AlertManager to store collected data and Grafana to visualize performance data.

C. Measurement Methodology

1) Experimental Testbed: We used $n1$-standard-8 instances on Google Compute Engine, with 8 Intel Broadwell vCPUs and 30GB of RAM, as the platform for each client and server. Each instance ran a minimal installation of Ubuntu 18.04.3 LTS using Linux 5.0.0-1021-gcp as the 64-bit kernel. We used Docker version 19.03.4 and Containerd version 1.2.10 hosting Nvidia’s TensorRT Inference Server version 1.6.0. cAdvisor version 0.33.0 and Node exporter version 0.18.1 were used to collect the server’s resource and performance information. We use version 1.6.0 Nvidia’s TensorRT Inference Framework Python Client SDK to perform inference requests using Python 3.6.8 on each client. The monitoring stack was composed of Prometheus version 2.11.1 and Grafana version 6.3.3. We chose to evaluate the GPU inference performance using Nvidia’s P4 and T4 GPUs due to their wide adoption, low price-point, and designation as data center inference products [33].

2) Model Selection: We used two popular CNN models, Inception-V3 [34] and ResNet50 [35], as the basis for evaluating the performance characteristics of inference. The models, which perform image classification tasks, require an image as input and produce a string as output. These two models were implemented in different frameworks: Inception-V3 uses TensorFlow [36] and ResNet50 uses

![Figure 1: Architecture of our measurement framework PERSEUS.](image)
Caffe2 [37]. This allowed for the use of original unmodified, pre-trained models and guarantees isolated model runtimes. It also demonstrated the framework-agnostic approach of PERSEUS.

3) Workloads: In our experiment we opted for a dataset of 6908 images, which was a randomly selected subset of the Open Images V3 Validation Dataset [38]. The images were preprocessed before inference to eliminate the overhead of loading and processing images at runtime. Inception-V3 requires 299 by 299 pixels RGB images and ResNet50 224 by 224 pixels RGB images. The dataset’s size provides the advantage of reducing the effects of abnormalities on results while maintaining a short runtime. Our framework delegates batching to the server where the server could treat each request as a single request from a client. The batch size determines the latency and throughput of the server. Therefore, we used the same batch size across all hardware configurations to provide a fair comparison.

4) Metrics: We evaluated the efficacy of the cost and performance tradeoff of multi-tenant model serving using PERSEUS framework, models, and workloads. We conducted several experiments to show the effect of hosting an additional model on startup time, latency, and throughput. Our key goal is to understand whether overhead introduced during multi-tenancy significantly impacts performance of inference. Peak or steady-state throughput λ measures the maximum rate of inference requests over a time span. The latency of requests t denotes the end-to-end response time for an inference request, where the 95th percentile latency is the latency for 95% of requests. Cost per one million inference requests c provides a standardized metric which promotes price comparisons across server hardware by accounting for the relative performance of a device [18].

More broadly, the measurement statistics are utilized to estimate the performance metrics for each stage of inference serving in regard to the hardware used. The startup time of various hardware platforms measures the time required to begin loading a model for inference on pre-provisioned server instances. The dedicated model performance is calculated by measuring the maximum computation capacity of a server under peak throughput λ, thus determining the stable operating range of a model on a given configuration. Finally, the multi-model performance is determined by measuring the resulting latency and throughput of each model served. The overhead and tradeoff of hosting multiple models is conveyed through the shift in peak workload performances and each model’s performance relative to its counterparts.

IV. PERFORMANCE AND COST CHARACTERIZATION

Utilizing PERSEUS, we evaluated on various serving options for deep learning inference. In practice, CNN models have been widely deployed and served with CPU only [27–29]. In this section, we first evaluate and study the tradeoffs of inference using CPUs versus GPUs. We then characterize the benefit of multi-tenant model serving with GPUs by comparing against dedicated GPU inference.

A. CPU vs. GPU inference

We first quantify the inference performance of two popular Convolutional Neural Networks with comparable model sizes, number of parameters, and inference accuracy (i.e., InceptionV3 and ResNet50) to demonstrate the importance and challenges of determining the appropriate serving hardware given the workload i.e. CNN models. Table III summarizes the inference time (batch size of 1) when executed on the CPU or a discrete GPU. When executed on the CPU, we define a hit to mean that the model was already present in main memory (when executing on the CPU) or in GPU memory (when executing on the GPU). A miss requires either loading models from the disk into the main memory or from the disk to GPU memory.

|                  | n1-standard-8 CPU | Nvidia P4 GPU | Nvidia T4 GPU |
|------------------|-------------------|---------------|---------------|
| Inception-V3     | Hit 159.1 ± 3.4    | 18.5 ± 0.6    | 18.2 ± 0.1    |
|                  | Miss 1401.4 ± 89.9 | 498.5 ± 310.6 | 21264.4 ± 310.6 |
| ResNet50         | Hit 75.1 ± 1.2     | 217.7 ± 1.8   | 325.9 ± 7.3   |
|                  | Miss 3806.7 ± 222.5| 18704.8 ± 343.4| 21663.3 ± 763.7 |

Table III: Average time (t) to perform inference for CPU versus GPU hardware. A hit means the model was already present in main memory (when executing on the CPU) or in GPU memory (when executing on the GPU). A miss requires either loading models from the disk into the main memory or from the disk to GPU memory.

Figure 2: Measurement illustration for CPU vs. GPU inferences.

We make three key observations. First, static model characteristics, such as model file size, are not a good indicator of runtime requirements and performance. Second, it is not always faster to execute the model on a GPU, even with GPUs optimized for inference such as NVIDIA’s P4 and T4 GPUs. For example, in the case of Inception-V3 (hit), it is more than three times faster to execute using an Intel CPU than the T4 GPU. However, we measured the peak throughput of both GPUs to be 12 times higher than that of the CPU with a batch size of 8. Third, even though the on-disk sizes of these two models are roughly the same, it takes twice as long to load Inception-V3 into CPU memory but
nearly the same amount of time to transfer each model from CPU to GPU memory. Our measurements both demonstrate the intricate trade-offs between caching in CPU memory versus GPU memory and motivate the need to mask the data transfer latency to the GPU memory.

Table IV shows the performance evaluation of CPU-based inference using same hyper-parameters as the GPU inference (i.e. a single model instance with a batch size of 8). Under steady-state conditions, the cost of performing one million inference requests at peak throughput on an 8-core CPU is significantly higher than GPU based inference under peak throughput conditions shown in Table V. The much worse performance of ResNet50 when serving batched requests is largely due to inference framework’s limited support for CPU inference. Specifically we observe that Caffe2 accumulated and processed batched requests on a single core. This suggests the need to carefully choose inference frameworks that are optimized for the underlying hardware [27, 39, 40]. Therefore, while CPU-based inference may be able to swiftly adapt to transient load spikes, it is not a cost and performance effective solution for handling workloads demanding higher throughput.

**Summary:** Serving with a cold cache is always better on CPU servers due to the high data transfer latency between CPU memory and GPU memory. Inference model miss incurs a time cost overhead ranging between 67X to 1168X compared to model hit on GPUs. While on CPUs, the overhead of model miss is at most 51X. However, some models are better suited for GPU serving with warm cache. For example, ResNet50 model on hit is up to 14X faster on GPU than on CPU, which does not hold true for InceptionV3.

**B. CNN inference on GPU**

1) Characterization of Dedicated Model Inference on GPUs: We profiled each model on available hardware configurations to establish the baseline performance for GPU based inference. Table V shows the dedicated model inference serving results using different GPU types and counts. Across both GPU types, the cost per inference and latency decrease when the number of GPUs increases. Accordingly, two GPU instances achieved higher overall throughput compared to the single GPU instances. On average, by adding an additional GPU, the price per inference request decreased by 14.43% for ResNet50 and 14.00% for Inception-V3. Surprisingly, for both CNNs, increases in peak throughput λ generated by T4 GPUs yielded a higher cost. Furthermore, running on a single P4 GPU Inception-V3 achieved a peak GPU utilization of 92% compared to ResNet50’s utilization of 88%. More importantly, on the GPU memory, utilization of Inception-V3 was 97.20% compared to 21.58% for ResNet50. In scenarios where maximum utilization can be achieved, the Dual P4 GPU configuration achieves the best cost-performance outcome.

Our data shows that under peak loads, GPU resources are under-utilized, i.e. cannot be fully leveraged, in multi-tenant inference. This suggests that performing multi-tenant inference on P4 GPU will lead to over-utilized GPU memory, due to the large size of each model in the card’s memory. As shown in Figure 3, in conditions where the peak throughput is not met, the cost increases polynomially as the throughput decreases. The results show that in scenarios of under-utilization, optimizing the hardware cost requires accurately estimating a model’s throughput. This in turn calls for understanding and modeling the characteristics of CNN models on how under-utilization can be effectively mitigated when serving another model to utilize the remaining resources.

While our experiment results suggest that testing a server instance with four GPUs may produce additional cost savings, we encountered issues when testing this configuration. When the four P4 or T4 GPUs were configured, the system became unstable. Subsequently, the data collected for the peak workload λ and latency t did not prove reliable. Our tests determined that the peak throughput for ResNet50 on four P4 GPUs achieved between 110% and 130% above the peak throughput of the equivalent two GPU servers. Ignoring the variability, the price per request at the maximum throughput did not justify the price of the server. In our experiments, the CPU, RAM, and GPU utilizations did not pose as the bottleneck. Additionally, the experimental findings of a Google Cloud Platform Blog article [41], which showed the average throughput of the network to be 10 GB/s, supports the assertion that our workloads of 0.5-1.0 GB/s did not result in a network bottleneck. The bottleneck was experimentally determined to be server’s gRPC endpoint.
Table V: Inference performance and cost with P4 and T4 GPUs. Serving with two P4 GPUs can be 18.4% cheaper for 1 million requests due to CPU cost amortization and linear throughput scalability. Increasing the number of T4 GPUs from 1 to 2, decreases the cost and latency of inference and increase the peak throughput across both models, results in 37.7% cost saving.

| Model            | One P4 GPU | Two P4 GPUs | One T4 GPU | Two T4 GPUs |
|------------------|------------|-------------|------------|-------------|
|                  | c ($/h)    | λ (reqs/sec) | c ($/h)    | λ (reqs/sec) | c ($/h)    | λ (reqs/sec) | c ($/h)    | λ (reqs/sec) |
| ResNet50 0.938 0.076 203.810 0.773 0.059 398.150 1.012 0.077 256.088 0.898 0.048 494.608 |
| Inception-V3 1.235 0.102 154.801 1.008 0.080 305.199 1.249 0.061 207.44 1.129 0.059 393.311 |

Table VI: Comparison of the lowest effective unit cost of multi-tenant model serving to server unit cost.

| Model            | One P4 GPU | Two P4 GPUs | One T4 GPU | Two T4 GPUs |
|------------------|------------|-------------|------------|-------------|
|                  | α ($/h)    | β ($/h)    | Savings (%) |            |
| ResNet50 0.688 1.108 0.933 1.598 |
| Inception-V3 0.753 1.241 1.026 1.754 |

Summary: For our selected workload on dedicated GPU server for inference, it is better to have more than one GPU card to share the workload, in terms of cost per request, when the request rate is higher than 100 requests/sec. Furthermore, even with a single GPU card, the GPU memory utilization for some models is underutilized, which could potentially be improved by serving multiple models on the same server.

2) Multi-Tenant Model Serving: To quantify the benefit of multi-tenant model serving, we evaluated the peak throughput, 95th percentile latency, and serving costs when the two CNN models share the underlying resources. Figure 4 compares the achieved peak throughput of ResNet50-dominated and Inception-V3-dominated requests sharing underlying GPU(s) with Inception-V3 and (ResNet50) to those of serving these two CNNs on the corresponding dedicated GPU(s), respectively. We observe that a multi-tenant model serving with such extreme workload mixes (e.g., 1:20 ratio) can achieve comparable throughput to a dedicated single model serving with one GPU. However, in the case of two GPUs, the aggregate throughput of multi-tenant model serving slightly lagged behind.

To understand the interplay between different multi-tenant inference workloads, we repeated the above measurements by adjusting the ratio of model requests. Figure 5 shows the achieved throughputs of ResNet50 and Inception-V3. The results show that it is a non-linear relationship as the requests for two models change. Thus, the overhead of hosting an additional model is less than the performance gain of exploiting under-utilized resources. This means hosting two models that cannot be both fully loaded onto a GPU’s memory does not make multi-tenant inference impractical. The performance gain occurs when the throughput is consistently achieved and both models are experiencing non-trivial workloads (i.e., between 25% and 75% of their peak throughput). Furthermore, we observe that the cost per inference and latency decrease when the number of GPUs increases, for both GPU types.

To quantify the relative cost saving of multi-tenant model serving with different workload mix ratios, we define a metric called effective unit cost. For a given server that costs \( a \) dollars per hour, if its capacity for serving ResNet50 is \( x \) requests per hour and for serving Inception-V3 is \( y \) requests per hour, then we can derive the server-model unit cost as \( \frac{x}{y} \). The effective unit cost is defined as \( b = \frac{ax}{x+y} \), where \( x' \) and \( y' \) are the number of requests the server can dedicate to serve ResNet50 and Inception-V3, respectively. Intuitively, \( a \) is the actual cost using multi-tenant model serving, and \( b \) describes how much one needs to pay for serving an aggregate request rate of \( x' + y' \). Therefore, the cost saving of multi-tenant model serving can be calculated as \( \frac{b-a}{a} \).

Table VI shows the results of performing the aforementioned calculations. When the request rates for both models converge to the same request rate, the effective unit cost is higher than that of a server hosting a single model. We show that across all hardware GPU configurations, it is on average 9.5% cheaper to serve the two models in this configuration. The steady-state latency for ResNet50 and Inception-V3 increased by 55% and 26% respectively. The results revealed that the best cost-performance ratio is achieved when both models are serving at the same request rate. In addition, serving multiple models can effectively achieve higher utilization of resources when a single model server experiences under-utilization.
Summary: Multi-tenant model serving can reduce the effective unit cost by up to 12% with two P4 GPUs. The maximum cost reduction for each hardware configuration was achieved when serving ResNet50 and Inception-V3 at roughly the same throughput. Our observations further suggest the benefits of intelligent provisioning and scheduling of inference requests using a multi-tenant approach.

V. CONCLUSION AND FUTURE WORK

As pre-trained deep learning models have been increasingly utilized for new application features and integrated into existing applications, it necessitates the research of resource-efficient inference serving. In this paper, we demonstrated the benefits of multi-tenant model serving, a promising way to improve server resource utilization and reduce monetary cost. We quantified its achieved performance and cost by comparing to other common serving configurations, using our measurement framework PERSEUS on Google Cloud. PERSEUS can also be easily leveraged to characterize the serving capacity for new CNN models and new hardware combinations. Through investigating and understanding the model serving performance, we further identified a number of performance bottlenecks, including inefficient framework supports for CPU inference and CNN model caching, that hindered the observed inference performance. Our study forms the basis for complementary research such as provisioning the inference servers and dispatching inference requests, which we plan to pursue as the next step.

ACKNOWLEDGMENT

The authors would like to thank National Science Foundation grants #1755659 and #1815619, and Google Cloud Platform Research credits.

REFERENCES

[1] C. Zhang et al., “Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving,” in 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019.
[2] A. Samanta et al., “No DNN left behind: Improving inference in the cloud with Multi-Tenancy,” arXiv:1901.06887, Jan. 2019.
[3] A. Jain, “Splitserve: Efficiently splitting complex workloads over iaas and faas,” 2019.
[4] V. Ishakian et al., “Serving deep learning models in a serverless platform,” in 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2018, pp. 257–262.
[5] Z. Tu et al., “Pay-per-request deployment of neural network models using serverless architectures,” in Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations, 2018, pp. 6–10.
[6] J. R. Gunasekaran et al., “Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud,” in 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019, pp. 199–208.
[7] A. Dakkak et al., “Trims: Transparent and isolated model sharing for low latency deep learning inference in function-as-a-service,” in 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019, pp. 372–382.
[8] A. Bhattacharjee et al., “Barista: Efficient and scalable serverless serving system for deep learning prediction services,” arXiv preprint arXiv:1901.01576, 2019.
[9] A. Gujarati et al., “Swayam: distributed autoscaling to meet slo of machine learning inference services with resource efficiency,” in Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. ACM, 2017, pp. 109–126.
[10] H. Qin et al., “Swift machine learning model serving scheduling: a region based reinforcement learning approach,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2019, p. 13.
[11] X. Tang et al., “Nanly: A qos-aware scheduling for dnn inference workload in clouds,” in 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2019, pp. 2395–2402.
[12] P. Jain et al., “Dynamic space-time scheduling for gpu inference,” arXiv preprint arXiv:1901.00041, 2018.
[13] P. Yu et al., “Salus: Fine-grained gpu sharing primitivies for deep learning applications,” arXiv preprint arXiv:1902.04610, 2019.
[14] S. S. Ogden et al., “MODE: Mobile deep inference made efficient by edge computing,” in USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.
[15] G. Li et al., “Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge,” in International Conference on Artificial Neural Networks. Springer, 2018, pp. 402–411.
[16] S. S. Ogden et al., “MDInference: Balancing inference accuracy and latency for mobile applications,” in Proceedings of IEEE International Conference on Cloud Engineering (IC2E 2020), 2020.
[17] G. R. Gilman et al., “Challenges and opportunities of dnn model execution caching,” in Proceedings of the Workshop on Distributed Infrastructures for Deep Learning, 2019, pp. 7–12.
[18] D. Crankshaw et al., “Clipper: A low-latency online prediction serving system,” in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017, pp. 613–627.
[19] F. Romero et al., “InfaaS: A model-less inference serving system,” 2019.
[20] C. Olston et al., “Tensorflow-serving: Flexible, high-performance ml serving,” arXiv preprint arXiv:1712.06139, 2017.
[21] “Nvidia tensorrt inference server,” https://github.com/NVIDIA/tensorrt-inference-server, 2019.
[22] “Apache predictionio,” https://github.com/apache/predictionio, 2019.
[23] “Redisa,” https://github.com/RedisAI/RedisAI, 2019.
[24] “Ai platform,” https://cloud.google.com/ai-platform/, 2019.
[25] “Azure machine learning,” https://azure.microsoft.com/en-us/services/machine-learning-studio/, 2019.
[26] “Sagemaker,” https://aws.amazon.com/sagemaker/, 2019.
[27] Y. Liu et al., “Optimizing CNN model inference on CPUs,” in 2019 USENIX Annual Technical Conference (USENIX ATC 19).
[28] J. Soifer et al., “Deep learning inference service at microsoft,” in 2019 USENIX Conference on Operational Machine Learning (OpML 19), 2019, pp. 15–17.
[29] M. Zhang et al., “Accelerating large scale deep learning inference through DeepCPU at microsoft,” in 2019 USENIX Conference on Operational Machine Learning (OpML 19), 2019, pp. 5–7.
[30] “Amazon elastic inference,” https://aws.amazon.com/machine-learning/elastic-inference/, 2019.
[31] V. J. Reddi et al., “Mlperf inference benchmark,” arXiv preprint arXiv:1911.02549, 2019.
[32] H. Demirkan et al., “Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud,” Decision Support Systems, vol. 55, no. 1, pp. 412–421, 2013.
[33] “Nvidia tensorrt hyperscale inference platform,” https://www.nvidia.com/en-us/deep-learning-ai/solutions/inference-platform/hpc/, 2019.
[34] C. Szegedy et al., “Rethinking the inception architecture for computer vision,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.
[35] K. He et al., “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[36] “Tensorflow,” https://github.com/tensorflow/tensorflow, 2019.
[37] “Pytorch,” https://github.com/pytorch/pytorch, 2019.
[38] I. Krasin et al., “Openimages: A public dataset for large-scale multi-label and multi-class image classification.” Dataset available from https://github.com/openimages, 2017.
[39] K. Hazelwood et al., “Applied machine learning at facebook: A datacenter infrastructure perspective,” in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb. 2018, pp. 620–629.
[40] J. Park et al., “Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications,” arXiv:1811.09886, Nov. 2018.
[41] “5 steps to better gcp network performance,” https://cloud.google.com/blog/products/gcp/5-steps-to-better-gcp-network-performance?hl=ml, 2017.