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

Matthew LeMay
mlemay@wpi.edu
Worcester Polytechnic Institute
Department of Computer Science
Worcester, MA

Shijian Li
sli8@wpi.edu
Worcester Polytechnic Institute
Department of Computer Science
Worcester, MA

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

ABSTRACT

Deep learning models are increasingly used for end-user applications, supporting both novel features, such as facial recognition, and traditional features, such as 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. Under-utilized server resources brought about by dynamic workloads can influence provisioning decisions, 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 various serving deployments and demonstrated that multi-tenant model serving can lead to up to 12% cost reduction.

KEYWORDS

DNN Inference, Multi-Tenancy, Performance Measurement

1 INTRODUCTION

Cloud Computing, namely Infrastructure as a Service (IaaS), has emerged as a popular platform for training and deploying deep learning models, due to their pay-as-you-go pricing models and 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 on various cloud hardware has garnered interest from researchers [32, 38].

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 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. In doing so, it is possible to decrease serving costs by multiplexing CNN models on previously underutilized servers.

In this paper, we show that multi-tenant model serving can achieve higher resource utilization and lead to promising cost savings, without violating performance guarantees for CNN models. Leveraging our measurement infrastructure called PERSEUS, we quantify 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, aiming for better resource utilization, evaluated the use of Functions-as-a-Service to handle transient workload scaling or as a replacement for IaaS [12, 14, 17, 20, 21, 36, 38], by eliminating over-provision and simplifying the scaling process. Other works have explored the use of predictive scaling [16], Quality of Service (QoS) aware scheduling [22, 29, 35], GPU primitive sharing [37], and edge-based techniques [24, 26] to improve serving resource 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 contribution to CNN inference serving research.

- Our empirical performance and cost characterizations of CNN model servings on different hardware configurations demonstrate the need for multi-tenant model serving, and up to 12% cost savings when appropriately mixing inference workloads.
- We design and implement a suite of tools, collectively referred to as PERSEUS[1], that can facilitate further evaluation of performance and cost trade-offs for new model serving

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] We will open source the project in GitHub. URL is omitted for anonymity.

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 1: *Overview of evaluated Nvidia GPU devices.*

deployment scenarios, such as running new CNN models on new 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.

The remainder of this paper is structured as follows: Section 2 introduces the key concepts underpinning CNN model serving systems and discusses previous studies done on the topic. Section 3 presents the problem statement followed by the design of Perseus and our measurement methodology for characterizing multi-tenant model serving, as presented in Section 4. Finally, we summarize the findings of our research and potential directions in section 5.

2 BACKGROUND AND RELATED WORK

There are numerous existing frameworks [4, 7, 9, 12, 13, 27, 31, 38] and services [2, 5, 10] 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.

2.1 Inference Serving Frameworks

Inference serving frameworks have evolved to support a wide array of use cases, libraries, and platforms. TensorFlow Serving [27] is one of the initial open-source inference serving systems that leverages GPUs. TensorFlow Serving supports multi-model deployments and exposes an endpoint for prediction, but requires models to be trained in the corresponding framework, TensorFlow. Other frameworks such as PredictionIO and RedisAI [4, 9] enabled the serving of models trained using different frameworks. Further, frameworks such as Nvidia's TensorRT Inference Server [7] provide hardware-specific inference optimizations, e.g., for Nvidia's GPUs.

Several frameworks have evolved to incorporate additional features aim to improve performance of inference serving. Clipper [13] adds additional functionality to ensure SLAs and to achieve better prediction accuracy. MArk [38] and Barista [12] leveraged Functions-as-a-Service (FaaS) to handle transient workloads while scaling in order to maintain SLAs. INFaaS [31] 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 scales each model independently of other models' resource footprints by setting GPU memory constraints. 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.

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 2: *Cloud server prices with various GPU configurations from Google Cloud* `us-central1-a`*, as of November 26th, 2019.*

2.2 Inference Serving Hardware

The abundance of commodity CPU servers in the cloud makes them ideal candidate for serving inference requests [25, 33, 39], 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 1 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.

2.3 Inference Serving Deployments

Deploying a CNN model to a pre-provisioned server requires developers to adhere to a given frameworks 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 [2, 4, 5, 7, 9, 10, 12–14, 27, 31, 38]. In systems that support autoscaling [12–14, 31, 38], middleware manages provisioning and acts as an single endpoint which routes requests to individual deployments.

Several major cloud providers offer managed inference serving frameworks such as Amazon's SageMaker, for deploying a single models in an isolated environment [2, 5, 10]. 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 [3], further reducing over-provisioning.

2.4 Inference Serving Performance Characterization

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

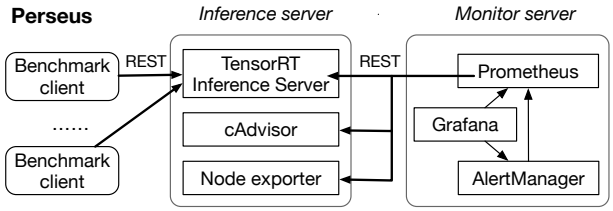

Figure 1: *Architecture of our measurement infrastructure* Perseus.

determines the viability of performing inference with a given configuration. SLA compliance is commonly measured by verifying that the 95^{th} or 99^{th} percentile of the end-to-end response time of recent requests is below a predefined threshold [16, 38]. The second-order goals of inference serving are throughput and cost. Deployments can require handling a large number of requests in a short time frame [33], thus accurately evaluating the throughput can help determine performance bottlenecks under heavy loads. Throughput is commonly measured by estimating the peak or steady-state request rate of the system [13, 30]. Table 2 shows the unit cost of performing inference on our choices of cloud server configurations.

3 PROBLEM STATEMENT AND MEASUREMENT METHODOLOGY

3.1 Problem Statement

In this paper, we investigate the performance and cost trade-offs of multi-tenant model serving compared to single-tenant serving as well as CPU serving. Such understandings can improve the resource efficiency of serving CNN models using cloud servers of various capacities. To do so, we designed and implemented a measurement infrastructure called Perseus which we then leveraged to quantify the model serving performance of various configurations. The configurations we explored include serving inference requests with CPU-only vs. with GPU hardware accelerator, as well as dedicated vs. shared GPU resources. Our measurements pinpoint several potential performance bottlenecks when serving CNN inference requests and demonstrate the cost savings prospect of multi-tenant model serving.

3.2 Perseus Architecture

In the past, there have been many works outlining the general procedures for determining the performance characteristics of an application on a server [15]. Due to the domain-specific intricacies of inference serving and application-specific constraints of working with an existing framework such as Nvidia's TensorRT Inference Server, we therefore propose a new measurement suite called Perseus. The new suite serves the purpose of gathering accurate performance data relevant to the server and models being served. We first describe the design and implementation of Perseus and then outline the models evaluated, the workloads used, and the experimental setup using the aforementioned framework.

Figure 1 shows the implementation of our performance measurement infrastructure. All components are encapsulated in Docker containers to ensure reproducibility. Perseus currently supports benchmarking image classification applications with Convolutional Neural Networks. The benchmarking client pre-processes the input data from a given directory, stores the pre-processed data in the client's memory, and then generates and sends inference requests to the inference server. The number of clients in conjunction with each client's estimation capacity determines the peak throughput (λ) 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 [7] and includes two additional components *cAdvisor* and *Node exporter* that aggregate and export the performance characteristics and resource utilization information—such as GPU, CPU and network utilization—of running containers. We further use *Prometheus* in conjunction with *AlertManager* to store all collected data and *Grafana* to visualize performance data.

3.3 Measurement Methodology

3.3.1 Experimental Testbed. We use 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. Docker version 19.03.4 and containerized version 1.2.10 hosted 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 Graphana version 6.3.3. We choose 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 [6].

3.3.2 Model Selection. We use two popular CNN models, `Inception-V3` [34] and `ResNet50` [19], 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 are implemented in different frameworks: `Inception-V3` uses TensorFlow [11] and `ResNet50` uses Caffe2 [8]. This allows for the use of original unmodified, pre-trained models and guarantees isolated model runtimes. It also demonstrates the framework-agnostic approach of Perseus.

3.3.3 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 [23]. Each image was scaled before beginning inference to the dimensions required by each model during pre-processing, in order to eliminate loading and processing time from the results. `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 size of the batch determines the latency and throughput of the server. Therefore, we use the same batch size across all hardware configurations to provide a fair comparison.

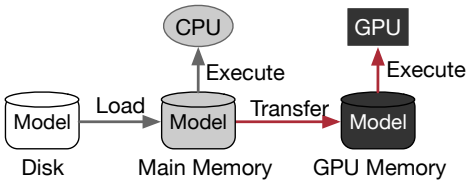

Figure 2: *Measurement illustration for CPU vs. GPU inferences.*

3.3.4 Metrics. We evaluate the efficacy of the cost and performance tradeoff of multi-tenancy using the profile framework, models, and workloads. We conduct 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 an indefinite time span. The latency of requests t denotes the end-to-end response time for an inference request, where the 95^{th} 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 [13].

More broadly, the measurement statistics are utilized to estimate the hardware-derived performance metrics for each stage of inference serving. The startup time of various hardware platforms measures the time required to begin loading a model for inference on pre-provisioned instances. Startup time ultimately determines the amount of resources used as a buffer for workload spikes. The single model performance is calculated by measuring the maximum 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.

4 PERFORMANCE AND COST CHARACTERIZATION

Utilizing Perseus, we conduct evaluation on various methods for deep learning inference. In practice, CNN models have been widely deployed and served with CPU only [25, 33, 39]. 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.

4.1 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` [34] and `ResNet50` [19]) to demonstrate the importance and challenges of determining the appropriate serving hardware given the workload. Table 3 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 (i.e., RAM) and a *miss* to mean that the model first needed to be loaded from disk. When executed on one of the two GPUs, we define a *hit* to mean that the model was already present in *GPU memory* and a *miss* to mean that the model needed to be loaded from disk into main memory and then transferred to GPU memory. Our measurement was conducted using Google Cloud, following the setup in Figure 2 and leverages our Perseus measurement infrastructure (Figure 1).

		Time to Execute on Device (ms)								
		n1-standard-8 CPU			**Nvidia P4 GPU**			**Nvidia T4 GPU**		
ResNet50	Hit	159.1	±	3.4	18.5	±	0.6	18.2	±	0.1
	Miss	1401.4	±	89.9	18418.4	±	498.5	21264.4	±	310.6
Inception-V3	Hit	75.1	±	1.2	217.7	±	1.8	325.9	±	7.3
	Miss	3806.7	±	222.5	18704.8	±	343.4	21693.3	±	763.7

Table 3: *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.*

	c ($)	**t_{95} (sec)**	**λ (reqs/sec)**
ResNet50	16.836	2.473	4.724
Inception-V3	4.029	0.765	19.720

Table 4: *Inference performance and cost with n1-standard-8. We configured a batch size of 8 and measured the cost c, 95^th percentile latency and peak throughput λ when serving 1 million requests.*

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` (model hit), it is more than three times faster to execute using an Intel CPU than the *T4 GPU*. However, we measure the peak throughput of both GPUs, where the batch size is 8, to be 12 times higher than that of the CPU. *Third,* even though the on-disk sizes of these two models is 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 tradeoffs between caching in CPU memory versus GPU memory and motivate the need to mask the data transfer latency to the GPU memory.

Table 4 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. 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 [18, 25, 28]. 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 of higher throughput.

Summary: Serving with a cold cache is always better on CPU due to 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`.

4.2 CNN inference on GPU

4.2.1 Characterization of Single Model Inference on GPUs. We profiled each model on available hardware configurations to establish the baseline performance for GPU based inference. Table 5 shows the single 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`. For both CNNs, increases in peak throughput λ generated by *Nvidia 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 Nvidia P4 GPU configuration achieves the best cost-performance outcome. Our data shows that under peak loads, GPU resources are underutilized. This suggests that performing multi-tenant inference on Nvidia P4 hardware 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 exponentially as the throughput decreases. The results show that in scenarios of under-utilization, optimizing the cost hardware requires accurately estimating a model's throughput. By accurately modeling a model's characteristics on hardware, under-utilization can be effectively eliminated when another model is served to utilize the remaining resources.

While our experiment results suggest that testing an 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. Under these conditions, the CPU, RAM, and GPU utilizations did not pose the bottleneck. Additionally, the experimental findings of a Google Cloud Platform Blog article [1], 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 represent a network bottleneck. The bottleneck was experimentally determined to be the gRPC endpoint of the server.

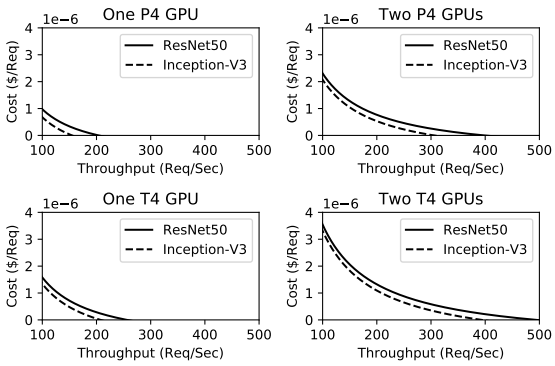

Figure 3: *The effects of throughput on inference cost showing the exponential increase in cost across all hardware configurations.*

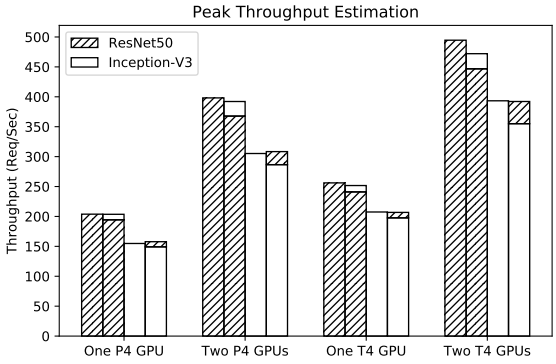

Figure 4: *Comparison of peak inference throughput for single vs. multi-tenant model serving. The first and third bars of each group represent the single dedicate model serving throughput, while the second and fourth bars describe the multi-tenant model counterparts.*

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/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 sharing the server.

4.2.2 Multi-Tenant Model Serving. To quantify the benefit of multi-tenant model serving, we evaluate the peak throughput, 95^{th} percentile latency, and serving costs when the two CNN models share the underlying resources. Figure 4 compares the achieved peak inference throughputs 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 a multi-tenant model serving lags behind.

	One P4 GPU			Two P4 GPUs			One T4 GPU			Two T4 GPUs		
	c ($)	t_{95} (sec)	λ (reqs/sec)	c ($)	t_{95} (sec)	λ (reqs/sec)	c ($)	t_{95} (sec)	λ (reqs/sec)	c ($)	t_{95} (sec)	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 5: *Inference performance and cost with P4 GPUs. Serving with two P4 GPUs can be 18.4% cheaper for 1 million requests due to CPU cost amortization and linear throughput scalability.*

	One P4 GPU	Two P4 GPUs	One T4 GPU	Two T4 GPUs
a ($/hour)	0.688	1.108	0.933	1.598
b ($/hour)	0.753	1.241	1.026	1.754
Savings (%)	9.45%	12.00%	9.96%	9.76%

Table 6: *Comparison of the lowest effective unit cost of multi-tenant model serving to server unit cost.*

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 there is not a linear relationship between both models. Thus, the overhead of hosting an additional is less than the performance gain of exploiting under-utilized resources. This means hosting two models that cannot 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 $\frac{a}{x}$ and $\frac{a}{y}$. The *effective unit cost* is defined as $b = \frac{ax'}{x} + \frac{ay'}{y}$ where x' and y' are the number of requests the server can service `ResNet50` and `Inception-V3`, respectively. Intuitively, b describes how much one needs to pay for serving an aggregate request rate of $x' + y'$ while the actual cost is a using multi-tenant model serving. Therefore, the cost saving of multi-tenant model serving can be calculated as $\frac{(b-a)}{a}$.

Table 6 shows the results of performing the aforementioned calculations. When the request rates for both models converge to 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 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 namely reveal that the best cost-performance outcome 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.

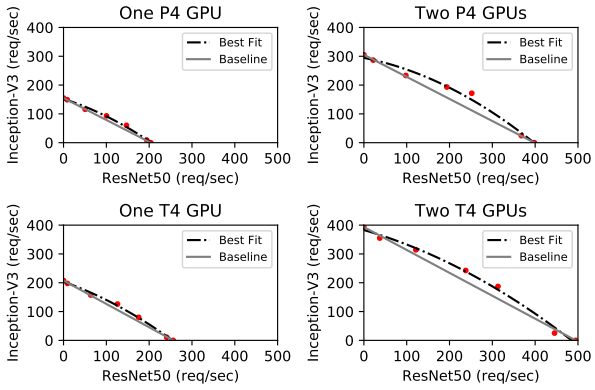

Figure 5: *Inference serving throughputs of multi-tenant model serving with different workload mix ratios.*

5 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, by quantifying its achieved performance and cost and comparing to other common serving configurations. Our empirical measurements on Google Cloud were enabled by Perseus, our measurement infrastructure. 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 also 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.

Acknowledgements. This work is supported in part by National Science Foundation grants #1755659 and #1815619, and Google Cloud Platform Research credits.

REFERENCES

[1] 2017. 5 steps to better GCP network performance. https://cloud.google.com/blog/products/gcp/5-steps-to-better-gcp-network-performance?hl=ml.

[2] 2019. AI Platform. https://cloud.google.com/ai-platform/.

[3] 2019. Amazon Elastic Inference. https://aws.amazon.com/machine-learning/elastic-inference/.

[4] 2019. Apache PredictionIO. https://github.com/apache/predictionio.

[5] 2019. Azure Machine Learning. https://azure.microsoft.com/en-us/services/machine-learning-studio/.

[6] 2019. NVIDIA TENSORRT HYPERSCALE INFERENCE PLATFORM. https://www.nvidia.com/en-us/deep-learning-ai/solutions/inference-platform/hpc/.

[7] 2019. NVIDIA TensorRT Inference Server. https://github.com/NVIDIA/tensorrt-inference-server.

[8] 2019. PyTorch. https://github.com/pytorch/pytorch.

[9] 2019. RedisAI. https://github.com/RedisAI/RedisAI.

[10] 2019. SageMaker. https://aws.amazon.com/sagemaker/.

[11] 2019. TensorFlow. https://github.com/tensorflow/tensorflow.

[12] Anirban Bhattacharjee, Ajay Dev Chhokra, Zhuangwei Kang, Hongyang Sun, Aniruddha Gokhale, and Gabor Karsai. 2019. BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services. *arXiv preprint arXiv:1904.01576* (2019).

[13] Daniel Crankshaw et al. 2017. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.

[14] Abdul Dakkak et al. 2019. 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, 372–382.

[15] Haluk Demirkan et al. 2013. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decision Support Systems* 55, 1 (2013), 412–421.

[16] Arpan Gujarati et al. 2017. Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM, 109–120.

[17] Jashwant Raj Gunasekaran et al. 2019. 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, 199–208.

[18] K Hazelwood, S Bird, D Brooks, S Chintala, U Diril, D Dzhulgakov, M Fawzy, B Jia, Y Jia, A Kalro, J Law, K Lee, J Lu, P Noordhuis, M Smelyanskiy, L Xiong, and X Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 620–629.

[19] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[20] Vatche Ishakian et al. 2018. Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 257–262.

[21] Aman Jain. 2019. SplitServe: Efficiently splitting complex workloads over IaaS and FaaS. (2019).

[22] Paras Jain et al. 2018. Dynamic Space-Time Scheduling for GPU Inference. *arXiv preprint arXiv:1901.00041* (2018).

[23] Ivan Krasin et al. 2017. OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages* (2017).

[24] Guangli Li et al. 2018. Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge. In *International Conference on Artificial Neural Networks*. Springer, 402–411.

[25] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. 2019. Optimizing CNN Model Inference on CPUs. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1025–1040.

[26] Samuel S Ogden et al. 2018. MODI: Mobile Deep Inference Made Efficient by Edge Computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

[27] Christopher Olston et al. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[28] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, Juan Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. *arXiv:1811.09886* (Nov. 2018). arXiv:cs.LG/1811.09886

[29] Heyang Qin et al. 2019. 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, 13.

[30] Vijay Janapa Reddi et al. 2019. MLPerf Inference Benchmark. *arXiv preprint arXiv:1911.02549* (2019).

[31] Francisco Romero et al. 2019. INFaaS: A Model-less Inference Serving System. arXiv:cs.DC/1905.13348

[32] Amit Samanta, Suhas Shrinivasan, Antoine Kaufmann, and Jonathan Mace. 2019. No DNN Left Behind: Improving Inference in the Cloud with Multi-Tenancy. *arXiv:1901.06887* (Jan. 2019). arXiv:cs.DC/1901.06887

[33] Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He, Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, Thomas Liu, and Junhua Wang. 2019. Deep Learning Inference Service at Microsoft. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. 15–17.

[34] Christian Szegedy et al. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[35] Xuehai Tang et al. 2019. Nanily: 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, 2395–2402.

[36] Zhucheng Tu et al. 2018. 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*. 6–10.

[37] Peifeng Yu et al. 2019. Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications. *arXiv preprint arXiv:1902.04610* (2019).

[38] Chengliang Zhang et al. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*.

[39] Minjia Zhang, Samyam Rajbandari, Wenhan Wang, Elton Zheng, Olatunji Ruwase, Jeff Rasley, Jason Li, Junhua Wang, and Yuxiong He. 2019. Accelerating Large Scale Deep Learning Inference through DeepCPU at Microsoft. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. 5–7.