Reconciling High Accuracy, Cost-Efficiency, and Low Latency of Inference Serving Systems

Mehran Salmani*, Saeid Ghafouri§, Alireza Sanaee§, Kamran Razavi†,
Max Mühlhäuser†, Joseph Doyle§, Pooyan Jamshidi‡, Mohsen Sharifi*

Iran University of Science and Technology*, Queen Mary University of London§,
Technical University of Darmstadt†, University of South Carolina‡

ABSTRACT

The use of machine learning (ML) inference for various applications is growing drastically. ML inference services engage with users directly, requiring fast and accurate responses. Moreover, these services face dynamic workloads of requests, imposing changes in their computing resources. Failing to right-size computing resources results in either latency service level objectives (SLOs) violations or wasted computing resources. Adapting to dynamic workloads considering all the pillars of accuracy, latency, and resource cost is challenging. In response to these challenges, we propose an adaptation mechanism, InfAdapter, that proactively selects a set of ML model variants with their resource allocations to meet latency SLO while maximizing an objective function composed of accuracy and cost. InfAdapter decreases SLO violation and costs up to 65% and 33%, respectively, compared to a popular industry autoscaler (Kubernetes Vertical Pod Autoscaler).

1 INTRODUCTION

The computing demand for machine learning (ML) has exponentially increased over the past decade [11]. For example, different ML applications, including computer vision, machine translation, chatbots, medical, and recommender systems, are running in data centers [13, 27, 30, 32], comprising more than 90% of computing resources allocated to ML [10, 13, 24]. ML inference services are user-facing, which mandates high responsiveness [17, 35]. Moreover, high accuracy is of great importance for these services [19, 25]. Consequently, inference systems need to deliver highly accurate predictions with fewer computing resources (cost-efficient) while meeting latency constraints under workload variations [17, 18, 35].

The dynamic nature of inference serving workloads requires different resource allocations for ML services [17, 35]. Failing to do so results in over or under-resource provisioning. Under-provisioning leads to service level objective (SLO) violations (*e.g.,* 99^{th} percentile of latency distribution, P99-latency)[17, 34]. Conversely, over-provisioning wastes computing resources [28, 34]. To address these problems caused by dynamic workloads, Auto-scaling [2, 9, 16, 17, 28, 34] resizes the resources of the service, and Model-switching [25, 36] switches between ML model variants that differ in their inference latency and accuracy (higher accuracy, higher latency); The former tries to be cost-efficient, and the latter tries to be more accurate, while both guarantee latency SLOs.

Table 1: InfAdapter is superior compared to the state-of-the-art solutions. (∗) Cocktail uses model ensembling leading to cost inefficiencies in particular scenarios (see Section 6).

Feature	MS [36]	INFaaS [28]	Cocktail [19]	VPA [9]	InfAdapter
Cost Optimization	✗	✓	✓*	✓	✓
Accuracy Maximization	✓	✗	✓	✗	✓
Predictive Decision-Making	✓	✗	✓	✓	✓
Container as a Service (CaaS)	✗	✗	✗	✓	✓
Latency SLO-aware	✓	✓	✓	✗	✓

Auto-scaling and model-switching as the state-of-the-art adaptation mechanisms fail to consider the accuracy and cost-efficiency simultaneously. Auto-scaling may sacrifice accuracy if it works with a low-accuracy model variant, or incur high resource costs if used for a high-accuracy model variant. Conversely, model-switching can be a subject of under-provisioning in cases where even the least accurate model variant is unable to respond to the workload; It also fails to be cost-efficient when the capacity of the most accurate model variant is more than the workload on the service.

The ability to jointly resize and switch ML model variants provides new opportunities. For instance, our experiments demonstrate that a Resnet50 model variant on 8 CPU cores allocation can sustain almost the same load that a Resnet152 variant does with 20 CPU cores; Moreover, a Resnet18 with 8 CPU cores can process the same load as a Resnet50 with 20 cores, while meeting P99-latency (750ms). Using a set of model variants instead of a single variant provides more granular accuracy/cost trade-offs.

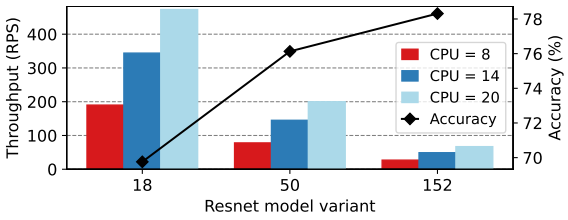

Figure 1: Throughput of three Resnet variants under 8, 14, and 20 CPU cores. We ensured the latency of all configurations is lower than 750 ms at the 99^{th} percentile at the saturation load.

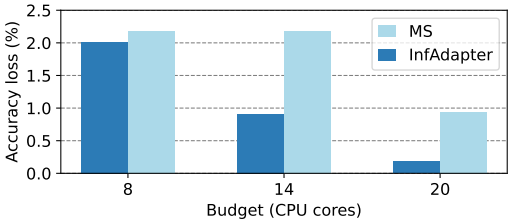

Figure 2: Comparison of InfAdapter (the ability to use a set of alternative models in the back-end) and Model-Switching+ on accuracy loss (accuracy of the most accurate model, Resnet152, subtracted by the accuracy of each bar). Each bar sustains the SLO of 750ms at P99-latency for a 75RPS load.

Inference systems should be adaptive in response to dynamic workloads and be able to consider all the *contrasting* objectives, including responsiveness (latency), accuracy, and cost-efficiency (allocated CPU cores), when dedicating resources to services and models. Moreover, reconciling these three measures is challenging as achieving one causes a violation or sacrifice of the other, and finding a trade-off among these three is daunting.

In response to these challenges, we design InfAdapter and empirically show that it can address the limitations of existing solutions. It predicts the service workload to mitigate provisioning overhead and, by using the predicted load, selects a set of ML model variants and their sizes (CPU cores) as the service backends to meet latency SLO and to maximize an objective function composed of average accuracy, resource cost, and loading cost (Section 3). To process the incoming requests, we implemented a dispatcher that load balances user requests to the backend variants according to their capacity(Section 4). Our experiments demonstrate that InfAdapter reduces average accuracy loss for latency SLO of 750 ms at 99^{th} percentile up to 4% given the same load compared to existing solutions (Section 5).

We have prototyped InfAdapter[1] in a Kubernetes cluster and used TensorFlow Serving [26] model server to serve our ML models. We experimentally evaluate InfAdapter using a real workload trace, Twitter-trace [12] (Section 5) and compare it against existing solutions (*e.g.*, vertical Pod autoscaler (VPA) [9], and Model-Switching [36]). Our experiments illustrate that InfAdapter reduces SLO violations by up to 65% compared to existing solutions. We further open-sourced our implementation for community engagement and reproducibility of our experiments. Table 1 summarizes differences between InfAdapter and other existing approaches.

2 MOTIVATION

Due to interaction with online users, inference services are latency-sensitive [17, 34], and since they contain heavy computations, they are resource-intensive [10]. Accuracy is also a pillar dimension of these services [25]. Faced with dynamic workload [17, 35], it is essential to consider the ternary trade-off space between latency, accuracy, and the resource cost in an adaptive way to address latency requirements while gaining higher accuracy cost-efficiently.

The importance of having accurate predictions while being cost-efficient, on one hand, hinders us from selecting a computationally light model variant with a low accuracy; On the other hand, selecting the most accurate model requires a very high resource cost to fulfill latency SLOs, which may even be unavailable. We conducted experiments with Resnet model variants under different CPU core assignments and captured their sustained throughput (number of requests they could handle given 750ms P99-latency SLO). Figure 1 shows our experiment's result for Resnet18, Resnet50, and Resnet152 variants used for image classification under 8, 14, and 20 CPU core assignments. In the figure, a Resent18 with 8 CPU cores, and a Resnet50 with 20 CPU cores, can almost sustain the same throughput under the latency SLO; A similar argument is applicable to Resnet50 with 8 cores and Resnet152 with 20 cores. Due to the fact that latency-accuracy trade-off space changes based on the workload, it is a non-trivial task to pick the right model variant with the right resource allocation.

> **Observation 1**. ML model variants provide the opportunity to reduce resource costs while adapting to the dynamic workload.

Using the traces collected from the previous experiment, we used two approaches to select backend model variant(s) to sustain a 75RPS load under a 750ms P99-latency SLO, using different CPU budgets (8, 14, and 20 CPU cores). Approach 1 is to opportunistically select a set of model variants and their sizes (InfAdapter). Approach 2 is to select a model variant and its size (MS). We compared the two approaches' accuracy loss (i.e., the accuracy we obtained subtracted from

[1]https://github.com/reconfigurable-ml-pipeline/InfAdapter

the accuracy of our most accurate variant, Resnet152). In Figure 2, we observe that InfAdapter is able to gain higher average accuracy (lower accuracy loss) for the requests by having more options to select from, i.e., selecting a set of models rather than a single model.

Observation 2. Using a set of model variants at the same time can provide better average accuracy compared to having one active variant.

Given dynamic workloads, we propose an adaptive mechanism for ML inference services to achieve latency SLO-aware, highly accurate, and cost-efficient inference systems. InfAdapter selects a subset of model variants to meet latency SLOs and maximizes an objective function of accuracy and cost. InfAdapter reconciles three important yet contradictory objectives (*accuracy, cost, and latency*).

3 PROBLEM FORMULATION

We formally describe the accuracy-cost problem using ML model variants while guaranteeing the latency SLO.

We denote M as the set of model variants for a specific task, with the latency SLO, L, the given accuracy of model $m \in M$, acc_m, and the model readiness time (loading the model into memory and the model initialization), rt_m. We profile our set of model variants under different CPU assignments to capture the number of requests they can process with respect to latency SLO L. Furthermore, by using the profiled data, we train a linear regression model to estimate the processing latency and throughput of model variant $m \in M$ under any CPU cores $n_m \leq B$, $p_m(n_m), th_m(n_m)$, where B is the total CPU budget and the total resource cost as $RC = \sum_{m \in M} n_m$.

To maintain system stability during a dynamic workload, the aggregated throughput of all available models for a given task must stay within an expected request rate λ. Mathematically, this can be expressed as $\sum_{m \in M, n \leq B} th_m(n_m) \geq \lambda$. Moreover, we define the weighted average accuracy, based on the quota of the workload on variant m, λ_m, as $AA = \sum_{m \in M} \frac{\lambda_m}{\lambda} \cdot acc_m$. Furthermore, we define the model loading cost as $LC = \max\{tc(m) * rt_m,\ m \in M\}$ where the transition cost, $tc(m)$, is equal to 1 if the model variant m needs to be loaded, and 0 otherwise. Table 2 summarizes the notations we use in the paper.

We define a multi-objective optimization problem to decide which subset of model variants to use such that under a given workload, the end-to-end latency is guaranteed. The goal is to maximize the weighted average accuracy AA and to minimize the total resource RC and loading LC costs. The problem can be formulated with the following integer linear programming (ILP):

Table 2: Notations

Symbol	Description
M	Set of all model variants for a given task
L	Latency SLO
m	An ML model variant from set M
acc_m	Accuracy of variant m
rt_m	Readiness time of variant m
tc_m	Transition cost of variant m
n_m	Number of CPU cores for variant m
$p_m(n_m)$	Processing latency of variant m with n_m CPU cores
$th_m(n_m)$	Throughput of variant m with n_m CPU cores
AA	Average Accuracy
RC	Resource cost
LC	Loading cost
B	CPU budget
λ	Workload on the system
λ_m	Workload quota on variant m

$$
\begin{aligned}
\max \quad & \alpha \cdot AA - (\beta \cdot RC + \gamma \cdot LC) \\
\text{subject to} \quad & \lambda \leq \sum_{m \in M} th_m(n_m), \\
& \lambda_m \leq th_m(n_m) \\
& p_m(n_m) \leq L, \forall m \in M, \\
& RC \leq B, \\
& n_m \in \mathbb{W}, \forall m \in M.
\end{aligned} \tag{1}
$$

In the objective function, we introduce α, β, γ to normalize the resource and loading costs and give importance to the objectives based on user preference. The first two constraints ensure the system's stability, e.g., there are enough resources to support the incoming workload. The third constraint satisfies the latency SLO, while the last two constraints bound the CPU core per model to be non-negative and within the available resources in the system. We use the Gurobi optimizer [20] to solve the ILP in the above equation.

4 SYSTEM DESIGN

An overview of the InfAdapter architecture is demonstrated in Figure 3. The system consists of three major components (monitoring, adapter, and dispatcher). *Monitoring* keeps monitoring statistics about the distribution of request arrivals. *Adapter* is responsible for first predicting the next time-interval workload based on the workload history gathered from the monitoring component and then finding a set of model variants, their CPU cores, and their workload quota by solving the ILP in Equation 1. *Dispatcher* controls distributing the requests to the set of multi variants based on the models' workload quota provided by the *Adapter* component.

Monitoring. The monitoring demon is in charge of fetching the arrival rate from the dispatcher. Precisely, we get the

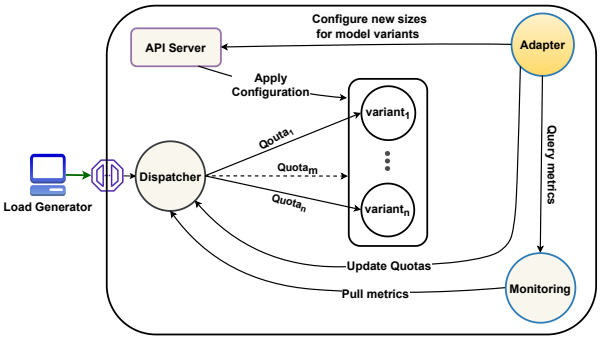

Figure 3: InfAdapter structure; Variants can be scheduled (by the Kubernetes scheduler) in any of the nodes.

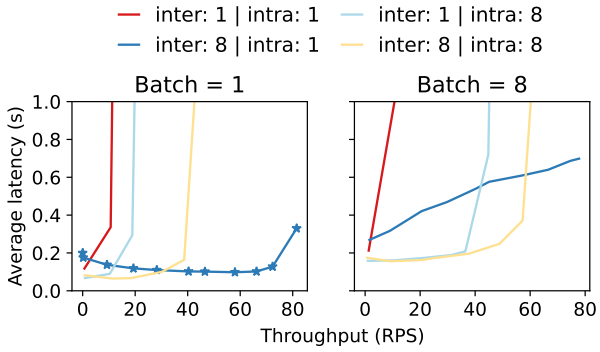

Figure 4: Throughput-Average latency for batch sizes of 1 (batching disabled) and 8 with different parallelism configurations on Resnet50 with 8 CPU cores allocations. The starred configuration is the chosen configuration through our experiments.

number of requests per second and pass it to the forecaster to predict the arrival workload for the next time interval.

Adapter. The Adapter consists of two sub-components, a time-series forecaster and a solver. We use LSTM [21] for time-series forecasting. Our LSTM model takes as input the load per second of the past 10 minutes, collected from the monitoring component, and predicts the maximum load for the next minute; We use the Twitter-trace dataset to train the LSTM model. Figure 5, top plot, shows the prediction accuracy of LSTM on a sample from the Twitter trace. The solver aims to solve the ILP in Equation 1 (every 30 seconds) to achieve the highest possible accuracy while respecting the latency SLO and available resources using the predicted workload and the current state of CPU allocation. Finally, the Adapter passes the set of models and their CPU cores to the cluster for enforcing the system configuration and the model's quota variables to the dispatcher for load balancing the incoming workload.

Dispatcher. The Dispatcher component load balances the incoming workload among the models in the cluster based on the received models' quota variable, λ_m, from the solver in the adapter component.

5 EVALUATION

We prototype InfAdapter in a Kubernetes cluster of two homogeneous physical machines from the Chameleon Cloud [23] equipped with 48 CPU cores of type Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz and 192 GiB of RAM. TensorFlow Serving is used to serve the model variants in separate Docker containers.

Batching and parallelism parameters. Batching and parallelism parameters are practical configuration knobs of ML inference services. Batching refers to aggregating multiple requests into one request, which is widely adapted for GPU inference systems [15, 22, 31, 33]. However, as shown in Figure 4, inference on CPU does not substantially benefit from

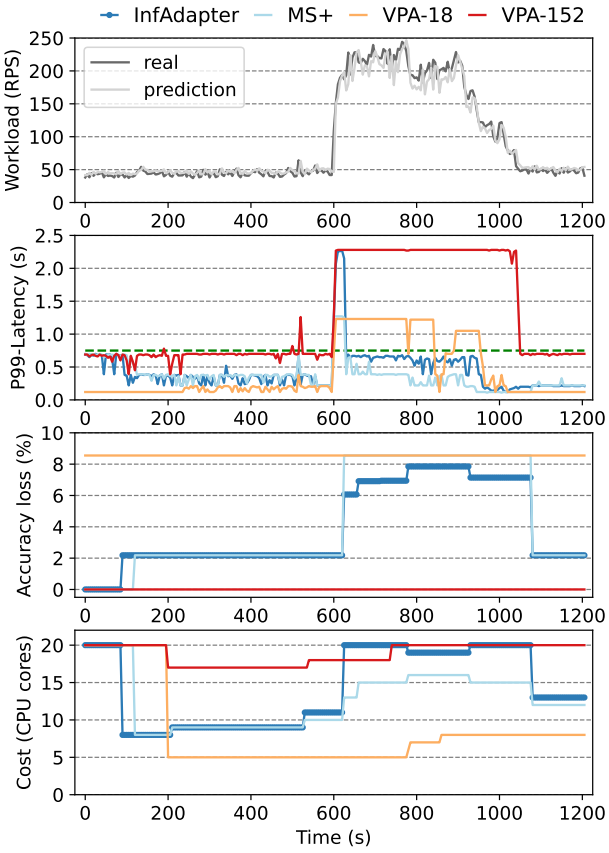

Figure 5: Comparison of InfAdapter with VPA used along with Resnet18, Resnet50 and Resnet152 on accuracy loss, cost and P99-latency, during the experiment, β = 0.05.

batching in terms of increasing the throughput, but increasing batch size leads to higher latency. Intra-op parallelism

defines the parallelism degree within an operation (such as matrix multiplication), and inter-op parallelism defines the parallelism across independent operations of inference requests [3–5].

We measured the effect of batching and CPU intra/inter operation parallelism on Resnet50 with 8 CPU cores in terms of throughput and latency. Experimental results are shown in Figure 4, which captures throughput-average latency relation under different batching and parallelism configurations. We choose (starred plot) to disable batching (set to 1), set inter-op parallelism to the number of CPU cores, and disable intra-op parallelism (set to 1) in InfAdapter across all the experiments to get the best throughput with a latency within the 750ms SLO. Further, we observed the same trend for all other model variants and CPU allocations.

InfAdapter handles bursty and non-bursty workloads. First, we experiment with bursty workloads to understand the performance of InfAdapter. We compared InfAdapter against an extended version of Kubernetes builtin Vertical Pod Autoscaler (VPA) [9], and an enhanced version of Model-Switching [36] (MS+). As the performance of the built-in VPA was very poor in the empirical evaluations, we made the following changes to it for a fair comparison against our approach. Initially, at each recommendation timestep, the builtin VPA removes the old container and then creates a new container with predicted resource allocations; This results in a service downtime during the recreation episode; To prevent this, we first create the container with the VPA recommended resources, and after it is up and running, remove the previous version. Secondly, We dropped the consideration of resource lower bound in VPA to scale up faster in response to the dynamic workload. For more information on the VPA algorithm, refer to [29]. Also, in MS+, since Model-Switching performs on a fixed resource budget, we add predictive allocation. At each time step, a model variant and its resource allocation are selected based on the same objective function we use for InfAdapter in Equation 1.

We evaluated the results on a 20-minute sample of Twitter-trace (Figure 5 top) that contains a steady load (0-600s), a load spike (600s-800s), a gradual decrease in the load (800s-100s) and a sample of going back to the initial load (1000s-1200s). It is evident that under the steady load, almost all the compared methods are able to stay under the 750 ms SLO. Once there is a load spike at 600s, almost all the compared methods suffer from SLO violations with a non-negligible margin (E.g., we observed a 10-minute violation for Resnet152). However, InfAdapter and MS+ temporarily trade-off a little accuracy in favor of being responsive to the load spike with a very short SLO violation time. Between MS+ and InfAdapter, InfAdapter is able to achieve the same SLO attainment with less accuracy loss during the load burst.

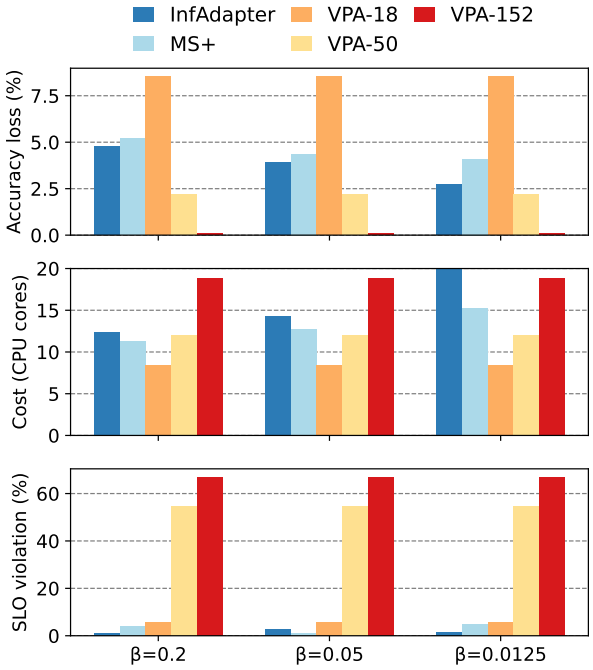

Figure 6: Comparison of InfAdapter with VPA used along with Resnet18, Resnet50 and Resnet152 on accuracy loss, cost and 99th percentile latency, for the whole experiment, under different β values.

InfAdapter aims to provide a flexible framework between accuracy and cost objectives. Under $\beta = 0.05$, we observed that InfAdapter is able to keep a balance between the cost and accuracy objectives and also comply with latency SLO. The same trend can be identified from the cumulative result of the entire experiment in Figure 6. The InfAdapter is always able to keep a better balance between the two cost and accuracy objectives compared to MS+. Also, VPA variants mostly took an extreme in maximizing only one objective. E.g., VPA-18 is the most cost-effective, but it comes at the expense of being very inefficient in accuracy.

Similarly, we used a non-bursty workload (Figure 7). We observed that InfAdapter has less accuracy loss compared to all other methods (except VPA+ with ResNet152, which has zero accuracy loss at the expense of high cost and SLO violations). Although in most cases InfAdapter has better SLO compliance, the difference between MS+ and InfAdapter in terms of cost and accuracy is small. We found that the difference was higher for a synthesized workload. In future work, we aim to test this with multiple workloads to examine InfAdapter's performance with different workloads.

6 RELATED WORK

Configuration of machine learning inference systems has gained considerable attraction in recent years. Clipper [15]

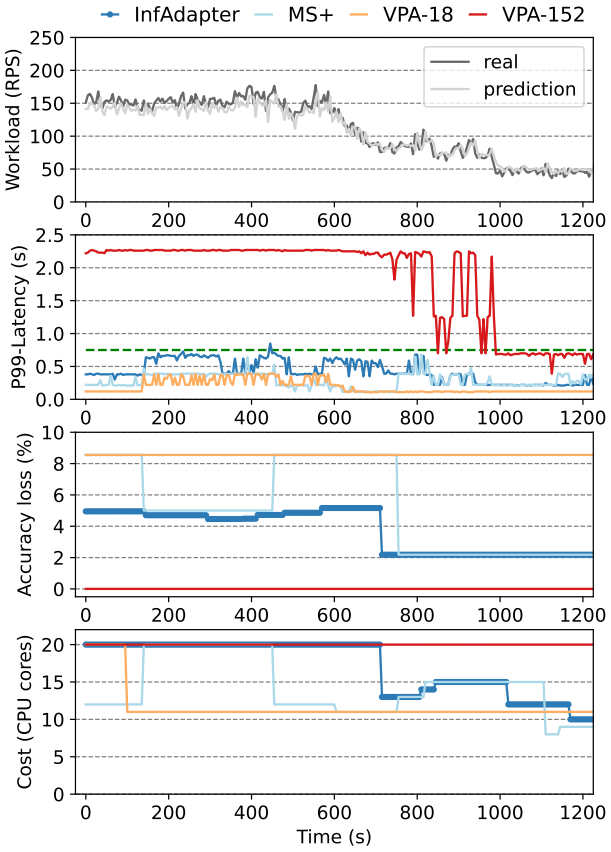

Figure 7: Comparison of InfAdapter with VPA used along with Resnet18, Resnet50 and Resnet152 on accuracy loss, cost and P99-latency, during the experiment with a non-bursty workload, β = 0.05.

is one of the early inference serving systems that introduced a general-purpose inference server with functionalities like caching, batching, and adaptive model selection.

MArk [34] employs request batching, predictive scaling, and serverless functions and proposes autoscaling policies that also take the hardware heterogeneity and service type diversity (FaaS, CaaS, IaaS) of inference serving data-centers into consideration.

INFaaS [28] provides an abstraction layer that decouples the model serving task from the used model for serving. Per each inference request, it searches through all the available sets of models for that specific inference task. Based on the request requirement, it finds the suitable model variant and dynamically offloads and unloads models as the user requirements change.

Model switching [36] is the first work that proposes switching between lightweight and heavier models as a workload adaptation mechanism. In order to be responsive to workload surges, it switches to a smaller but less accurate model. Unlike InfAdapter, their model is not cost-aware and can only work under a fixed resource budget.

Cocktail [19] is the most similar work to InfAdapter. It proposes an approach based on ensemble learning to reduce the cost while meeting the previous works' latency and accuracy efficiency. Cocktail uses ensembling as its accuracy maximization technique, which is costly as all the requests should be sent to all the ML models. Most of the time, a large number of model sets should be used to get to the accuracy of the largest model. Cocktail uses transient virtual machines to improve cost efficiency. Nevertheless, using unstable transient instances can cause interruptions in the inference service. Deploying InfAdapter on CaaS platforms like Google Autopilot does not suffer from similar problems. Due to fundamental structural differences and different problem formulations, we could not compare InfAdapter with Cocktail.

7 FUTURE WORKS

Hardware Heterogeneity. While in this work, we focused on homogeneous CPU inferencing, the performance of InfAdapter under general purposed (GPUs) and ASIC ML hardware can be evaluated. With packing requests into batches, GPUs can process higher workloads without a considerable increase in latency.

Scalability with ML. Our proposed solution works by brute-forcing through all possible configurations and picking the one that maximizes the objective function. Such an approach could suffer from scalability in case of growth in configuration space (more model variants and bigger resource budgets in our case). Utilizing ML-based solutions can decrease the amount of sampling in the search space, resulting in faster decision-making.

Multi Model Serving. In the case of using accelerators like GPUs, it is hard to share them among several containers, as there is no built-in mechanism for GPU sharing in container orchestration platforms like Kubernetes [1]. The multi-model deployment pattern, adapted in most production ML model servers [6–8, 14, 26], can mitigate the issues. Considering these emerging ML serving paradigms for improving adaptation mechanisms is a potential future work.

8 CONCLUSION

In this work, we presented InfAdapter, an adaptation mechanism for ML inference services. It selects a set of ML model variants and their resource allocations to achieve a trade-off between accuracy and cost while preserving latency SLO guarantee. Experiments on real-world traces showed that InfAdapter adapts better to dynamic workloads compared to the existing solutions by utilizing scaling and ML model variants selection.

REFERENCES

[1] 2022. GPU Virtualization in K8s: Challenges and State of the Art. (Nov 2022). https://www.arrikto.com/blog/gpu-virtualization-in-k8s-challenges-and-state-of-the-art/

[2] 2022. Horizontal Pod autoscaling. (Jun 2022). https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[3] 2022. Inter-op parallelism threads. (2022). https://www.tensorflow.org/api_docs/python/tf/config/threading/set_inter_op_parallelism_threads

[4] 2022. Intra-op parallelism threads. (2022). https://www.tensorflow.org/api_docs/python/tf/config/threading/set_intra_op_parallelism_threads

[5] 2022. TorchServe parallelism threads. (2022). https://pytorch.org/docs/stable/notes/cpu_threading_torchscript_inference.html

[6] 2023. Kserve. https://github.com/kserve/kserve. (2023).

[7] 2023. Seldon. https://github.com/SeldonIO/seldon-core. (2023).

[8] 2023. Triton inference server. https://github.com/triton-inference-server/server. (2023).

[9] 2023. Vertical Pod autoscaling. https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler. (2023).

[10] Sherif Akoush, Andrei Paleyes, Arnaud Van Looveren, and Clive Cox. 2022. Desiderata for next generation of ML model serving. *arXiv preprint arXiv:2210.14665* (2022).

[11] Dario Amodei and Danny Hernandez. 2019. AI and compute. https://openai.com/blog/ai-and-compute/. (Nov 2019).

[12] archiveteam. 2021. Archiveteam-twitter-stream-2021-08. https://archive.org/details/archiveteam-twitter-stream-2021-08. (2021).

[13] Jeff Bar. 2019. Amazon EC2 ML inference. https://tinyurl.com/5n8yb5ub. (Dec 2019).

[14] PyTorch Serve Contributors. 2020. Torch serve. https://pytorch.org/serve/. (2020).

[15] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: a low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 613–627.

[16] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. 2012. Autoscale: dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)* 30, 4 (2012), 1–26.

[17] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S McKinley, and Björn B Brandenburg. 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*. 109–120.

[18] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like clockwork: performance predictability from the bottom up. *arXiv preprint arXiv:2006.02464* (2020).

[19] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. 2022. Cocktail: a multidimensional optimization for model serving in cloud. In *USENIX NSDI*. 1041–1057.

[20] Gurobi Optimization, LLC. 2023. Gurobi optimizer reference manual. (2023). https://www.gurobi.com

[21] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[22] Yitao Hu, Rajrup Ghosh, and Ramesh Govindan. 2021. Scrooge: a cost-effective deep learning inference system. In *Proceedings of the ACM Symposium on Cloud Computing*. 624–638.

[23] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[24] George Leopold. 2019. AWS to offer Nvidia's T4 GPUs for AI inferencing. https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform/. (Mar 2019).

[25] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: timely inference serving for dynamic edge networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 277–290.

[26] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-Serving: flexible, high-performance ML serving. *arXiv preprint arXiv:1712.06139* (2017).

[27] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. 2018. Deep learning inference in Facebook data centers: characterization, performance optimizations and hardware implications. *arXiv preprint arXiv:1811.09886* (2018).

[28] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. {INFaaS}: automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[29] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, et al. 2020. Autopilot: workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[30] Devvi Sarwinda, Radifa Hilya Paradisa, Alhadi Bustamam, and Pinkie Anggia. 2021. Deep learning in image classification using residual network (ResNet) variants for detection of colorectal cancer. *Procedia Computer Science* 179 (2021), 423–431.

[31] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[32] Leonid Velikovich, Ian Williams, Justin Scheiner, Petar S Aleksic, Pedro J Moreno, and Michael Riley. 2018. Semantic lattice processing in contextual automatic speech recognition for Google assistant. In *Interspeech*. 2222–2226.

[33] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: fast, near-optimal auto-configuration for cloud-native model serving. In *Proceedings of the ACM Symposium on Cloud Computing*. 639–653.

[34] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*. 1049–1062.

[35] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and {delay-tolerance}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 377–392.

[36] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-switching: dealing with fluctuating workloads in machine-learning-as-a-service systems. In *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.