Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems

Jeff (Jun) Zhang
New York University

Sameh Elnikety
Microsoft Research

Shuayb Zarar, Atul Gupta
Microsoft

Siddharth Garg
New York University

Abstract

Machine learning (ML) based prediction models, and especially deep neural networks (DNNs) are increasingly being served in the cloud in order to provide fast and accurate inferences. However, existing service ML serving systems have trouble dealing with fluctuating workloads and either drop requests or significantly expand hardware resources in response to load spikes. In this paper, we introduce Model-Switching, a new approach to dealing with fluctuating workloads when serving DNN models. Motivated by the observation that end-users of ML primarily care about the accuracy of responses that are returned within the deadline (which we refer to as effective accuracy), we propose to switch from complex and highly accurate DNN models to simpler but less accurate models in the presence of load spikes. We show that the flexibility introduced by enabling online model switching provides higher effective accuracy in the presence of fluctuating workloads compared to serving using any single model. We implement Model-Switching within Clipper, a state-of-art DNN model serving system, and demonstrate its advantages over baseline approaches.

1 Introduction

Deep neural networks (DNN), currently the state-of-the-art in machine learning (ML), are being increasingly deployed as cloud services to provide highly-accurate inferencing (or predictions) for a range of applications. Systems such as Clipper [10] and Tensorflow serving [31] have been developed to ease the challenges in the deployment, optimization, and maintenance of DNN based machine-learning-as-a-service (MLaaS). Like other cloud services, MLaaS has quality of service (QoS) requirements in the form of service level agreements (SLAs) between the user and the cloud provider that provide guarantees on request latency, throughput and reliability. For ML, however, the *prediction accuracy* (or simply accuracy) of the model is also a critical metric but has not traditionally been encapsulated in SLAs.

In this paper, we make two observations to address this gap. First, we observe that for ML workloads, clients are interested not in the fraction of predictions returned within a deadline, but instead the fraction of *correct* predictions returned within the deadline. We refer to this metric as the *effective accuracy*; in Section 3, we show that the effective accuracy is the product of the ML model's accuracy and its deadline meet rate. *Second*, we observe that an SLA specified in terms of effective accuracy enables flexibility in dealing with spikes in workload, for instance, those observed during events like Black Friday.

The flexibility arises from the fact that deep learning models of varying computational complexity and accuracy can be trained for the same application. We observe that as load increases, the cloud provider can swap out complex and highly accurate models for more computationally efficient models while *preserving* effective accuracy. We refer to this approach as *Model-Switching*. From the cloud providers' perspective, *Model-Switching* allows to make meaningful trade-offs between the computational cost and the service accuracy. For instance, consider a web-serving application, which employs high-performance machine-learning models to recommend relevant items to users or show them ads. In such applications, whenever there is a spike in user load, cloud providers either throttle the serving rate of the application or scale-out computational resources to meet the demand and thus the latency SLA. However, in the former, there is a hit to the throughput SLA (or servable queries-per-second), and in the latter, there is an associated hardware cost for the cloud provider [1]. A third alternative to these options is to guarantee effective accuracy. With this approach, latency and throughput SLAs can always be met at the cost of limited accuracy loss. Without this knob, clients have to either give up on latency or throughput under spiking load.

To illustrate the benefits of the proposed approach, we develop and evaluate Model-Switching, an online scheduler built

[1] Google Cloud ML's documentation is reflective of the current approach: "If your traffic regularly has steep spikes, and if reliably low latency is important to your application, you may want to consider manual scaling [1]."

on top of *Clipper* [10] that monitors and adapts to workload fluctuations by switching between a set of pre-trained models for image classification. To further improve efficiency, Model-Switching also optionally determines the optimal number of threads and replicas for each model. Our evaluation shows that Model-Switching yields the highest *effective accuracy* for all deadline constraints compared with serving with each single model by itself.

The remainder of the paper is organized as follows: Section 2 provides a brief overview of deep learning and the limitation of existing MLaaS frameworks; the proposed effective accuracy, our new QoS metirc, and the model-switching framework are described in Section 3 followed by results from preliminary evaluation in Section 4. Section 5 describes related work while Section 7 lists the limitations of current approach and future work. Section 6 ends the paper.

2 Background and Motivation

We begin by briefly describing the deep learning inference and the limitations of existing MLaaS approaches.

2.1 DNNs and MLaaS

DNN Basics State-of-the-art DNNs are typically trained using GPU-enabled machine learning frameworks [7,24,32] like PyTorch, TensorFlow, or Caffe to obtain the model *weights*. Trained models can then be deployed into IoT and embedded devices for inferencing (i.e., to render predictions); in practice, state-of-the-art DNN models can be computationally demanding and hence inference is often outsourced to the *cloud*.

The execution time of DNN inference depends on its depth, the size of each layer's feature maps and filters. Fig. 1 shows the execution time for a family of ResNet models with varying depth (for example, ResNet-18 has 18 layers). From the figure, we can observe a more than 5× spread in execution times and that more complex, deeper models are also more accurate. (Also shown in the figure is the impact of thread-level parallelism on each model's execution time, which will be discussed later in Section 3.2.1).

MLaaS Framework Several DNN prediction serving systems have been built [10, 31, 39] to ease the deployment, optimization, and maintenance of DNN inference services. In these systems, DNN models are usually deploy into *containers*, or *servables*. State-of-art serving systems also enable model versioning, updates, rollbacks, replication, etc. At runtime, they enable caching, adaptive batching, and ensembling, together with *auto-scaling policies* [8, 9, 16–18, 30] to sustain QoS and manage hardware resources. Nonetheless, these systems have some drawbacks, described next.

2.2 Limitation of Exiting MLaaS Framework

Despite recent advances, state-of-the-art MLaaS frameworks still do not perform well in the presence of highly fluctuating loads or load spikes. Existing frameworks like *Swayam* [18] and *Clipper* [10] choose to violate SLAs in the presence of load spikes in order to conserve hardware resources. Both systems internally track each request's deadline in the queue and *prune* (or drop) the request if the queuing delay will result in a deadline miss. As a consequence, *Swayam* is only able to guarantee that 96% of requests return within the deadline during load spikes, even though the target SLA requires 99% of the jobs to meet deadline. In return, *Swayam* provides 27% resources savings compared to a baseline that scales hardware resources in response to load spikes. To summarize, existing MLaaS serving systems offer an unappealing trade-off for bursty workloads: either violate SLAs or incur significant hardware overheads. In this paper we show that this trade-off can be averted using the proposed model switching scheme and without the need to scale up resources.

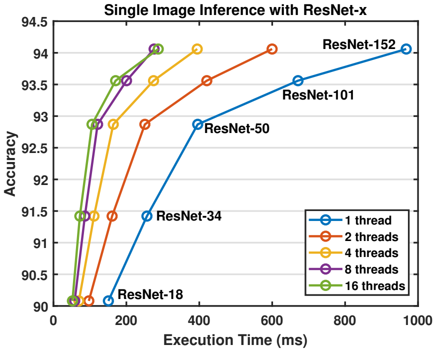


Figure 1: Inference time for ResNets in Pytorch [32] on a 32-vCPU host machine.

3 Model-Switching: Our Approach

We start with the new metric, i.e., *effective accuracy* for MLaaS systems, and then introduce our Model-Switching Framework.

3.1 Effective Accuracy

We argue that for MLaaS, response time (latency) alone does not capture end-users' expectation; instead, users are interested in both *fast* and *accurate* responses. To this end, we define **effective accuracy** within a deadline constraint as the *fraction of correct (or accurate) predictions returned within the deadline*. We will assume that users are agnostic to which model the service provider uses as long as the SLA, specified in terms of effective accuracy, is met.

Assume a pre-trained library of M ML models for given task where each model $i \in [1, M]$ is pre-characterized in terms of its accuracy a_i and $p_i^{D,\lambda}$, the fraction of requests that meet deadline D assuming requests arrive at rate λ. Then, the effective accuracy, a_i^{eff} is simply:

$$a_i^{eff} = p_i^{D,\lambda} * a_i. \quad (1)$$

Note that the execution time of a DNN is typically fixed and input independent; consequently, deadline misses are statistically independent of mis-classifications, thus enabling us to express the effective accuracy as a product of probabilities.

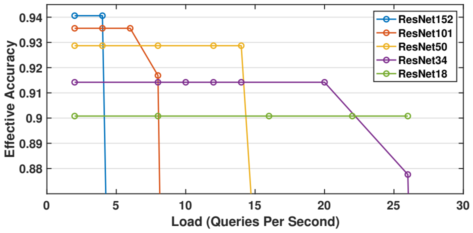


Figure 2: Effective accuracy as a function of increasing job arrival rate.

Figure 2 shows the effective accuracy for five ResNet models with increasing load (requests/second) assuming a deadline of 750 ms. At low loads, the most complex ResNet model, since it has the highest baseline accuracy and, since queuing delay is negligible under low loads, always meets the deadline. However, as load is increased, the simpler models have higher effective accuracy than more complex models because the latter incur a much higher fraction of deadline misses.

3.2 Online Model-Switching

Based on the observations above, our online model-switching framework monitors and predicts future job arrivals and switches between models to maximize effective accuracy. In addition, once a model is picked, the framework also selects the optimal number of threads and replicas of the model given the hardware constraints. We begin by discussing the impact of number of threads and model replicas on performance.

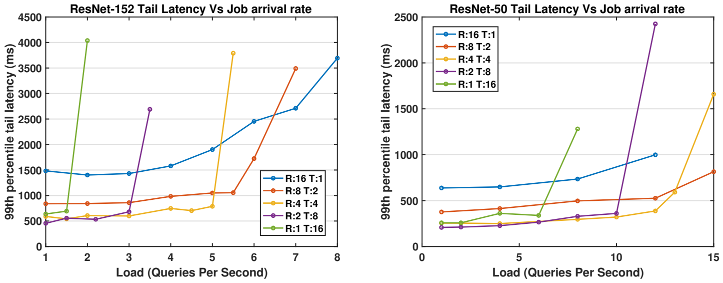
3.2.1 Job-level and Thread-level Parallelism

DNN model-based microservices, along with other general cloud computing workloads, offer variety of opportunities in terms of parallelism [12, 28]. In MLaaS setting, as requests queue up for processing, two decisions can be made to exploit the parallelism at different granularity: (1) How many requests can be serviced in parallel? The answer depends on the number of microservice **replicas** (R) we have in the system; (2) Once a request is assigned to one of the DNN models, how many **threads** (T) should be allocated to this microservice (as shown in Fig. 1)?

In this paper, we assume fixed capacity C of computing resources (i.e., CPU cores) for serving job requests. (If required, the proposed approach can be easily combined with auto-scaling frameworks that increase C in response to spikes in workload.) Under this assumption, any combination of $<R,T>$ that satisfies $R \times T = C$ can be chosen.

To understand the impact of the choice of $<R,T>$ on performance, we deploy several ResNet models in Clipper [10] and measure the end-to-end query latency by varying $<R,T>$ combinations at varying load levels. More information about Clipper and on the experiment set-up can be found in Section 4.

Fig. 3 summarizes the 99th percentile (P99) query latency for ResNet-50 and ResNet-152 models for five different $<R,T>$ configurations. It can be observed that judiciously pick $<R,T>$ is necessary; the optimal number of threads T reduces with increasing load. Qualitatively similar results hold for the other ResNet models but are not shown here due to space constraints.



(a) ResNet-152.

(b) ResNet-50.

Figure 3: Tail latency as job arrival rate increases.

3.2.2 Request Rate Prediction

In this paper, we use event-based windowing to monitor load at run-time, and use the load measured in a given window as a predictor for the next window. Clipper records each incoming request's arrival timestamp internally. Our Model-Switching controller estimates the inter-arrival rate by using the youngest and oldest timestamps with a fixed window size *periodically*. The window size can be tuned offline to improve the responsiveness. A similar approach was also used in [35].

3.2.3 Rule Based Model-Switching

With pre-characterized information about the P99 latency for each model, and the request-rate estimate, the Model-Switching controller searches over all M models and their $<R, T>$ configurations to pick the model and configuration that has the highest effective accuracy for the specified deadline. Note that doing so maximizes the chances that any given SLA constraint specified in terms of effective accuracy would be met. While our goal of maximizing effective accuracy might create some slack between offered and required service quality, this slack can be exploited by scaling *down* hardware resources (although we do not explore resource scaling in this paper).

Although our Algorithm 1 is general and able to explore dynamic $<R,T>$ allocation, we can see that $<R{:}4,T{:}4>$ works

Algorithm 1: SLA Deadline-Aware Model-Switching Plan Generation.

```
Data: A fixed capacity of CPU resources, $C$; target
      SLA deadline $D$; intermediate load, $Q$; a pool of
      candidate DNN models, $M$ (sorted with
      descending baseline accuracy).
Result: Model index and combination of <$R$,$T$> for
        online prediction serving.
$index \leftarrow M, R \leftarrow C$;
for $m \in [1, M]$ do
    $index \leftarrow m$;
    $< R, T > \leftarrow \arg\min_{<R,T>:R\times T=C} P99(m, Q)$;
    if $P99_{<R,T>}(m, Q) \leq D$ then
        return $index, < R, T >$;
    else
        Pass;
    end
end
return $index, < R, T >$;
```

effectively on ResNets across the board for most target deadlines, shown in Fig. 3. In our evaluations, we statically pick this configuration and do not consider this problem further in this paper.

3.2.4 Additional Considerations

Model Setup Times ML backend setup incurs large provisioning delays (e.g., a few seconds) due to massive I/O operations. To alleviate this issue, in this work, we pre-deploy all candidate models, relying on the fact that the RAM resources are usually abundant and under-utilized.

Similar ideas have also been proposed recently in Multi-Tenant Serving system aiming for better resource utilization [29, 33]. We also quantify the actual memory cost for hosting 5 models simultaneously in Section 4. More discussion is also available in Section 7.

CPU Resource Contention Hosting multiple DNN models may incur CPU performance overhead. However, at each given time, only one model would be required in active mode. To minimize performance impact, we reduce the CPU priority of the remaining "inactive" models via the OS level scheduler. We validated this approach and found that with this optimization, the performance is almost the same as only deploying a single model by itself.

4 Evaluation

We build our Model-Switching controller into Clipper [10], an open source online prediction serving system. To be focus on this study, we disable the cache and dynamic batch size adaption in Clipper.

System Configuration We use a dedicated Azure Virtual Machine (VM) with 32 vCPUs and 128GB of RAM for Clipper model serving and our Model-Switching controller. For the client, we have another separate VM (8 vCPUs and 32GB RAM) to send image queries. We set batch size of 1 when posting the request.

Inference Models We primarily look at deep residual nets (ResNets) [22] with various number of layers, baseline accuracy and execution time, shown in Fig. 1. Each model is pre-trained in Pytorch [32] on Imagenet [13], and deployed into *container* with <R:4,T:4> as microservices (discussed in Section 3.2.1) in Clipper. At any given time, only one model's containers are in the active state.

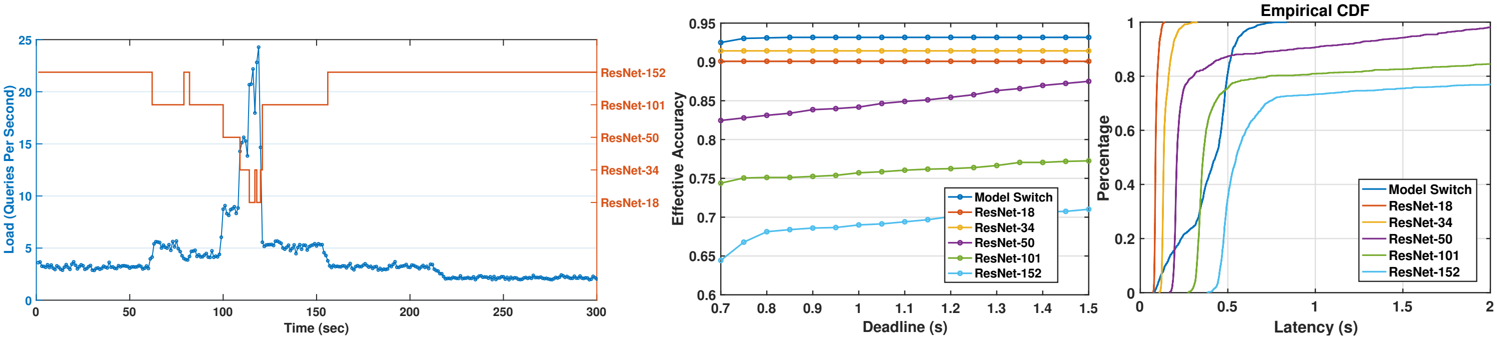
Workload Generator The load generator tries to emulate user behavior with a Markov model [11, 14]. It operates in an open system model [34], i.e., new jobs arrive independently of job completions and following Poisson inter-arrivals [11, 35]. We also validated the generated workload with a trace of job arrivals from a production system deployed in industry.

Model-Switching Controller The controller runs as part of the *Clipper* serving system with a sample period of 1 second and tracks the most recent incoming queries to the TASK QUEUE to measure load. It then determines and switches to the best model with a target SLA deadline of 750 *ms*. The deadline is selected to make sure the largest ResNet-152 is a feasible solution at low load.

4.1 Results

Fig. 4a shows the load profile (queries/second) over a 300-second period for the industrial trace and the models selected by the Model-Switching controller during this period. From the figure, we can see the the controller selects the most accurate but also computationally expensive ResNet-152 model when the load is relatively low. For moderate loads, the controller switches between ResNet-101 and ResNet-152. However, when the load spikes at the 125-second mark, the controller can quickly adapt and serves requests using the smaller ResNet-50, ResNet-34, or even ResNet-18 models. Note that all the switching happens in real-time, together with the prediction serving. The results account for all overheads of switching between models.

Effective Accuracy Fig. 4b shows the effective accuracy of the proposed Model-Switching controller compared to baselines that make use of a single model only. The results are shown for deadline constraints ranging from 700ms to 1500ms using the same workload trace from Fig. 4a. We observe that since both ResNet-18 and ResNet-34 are fast and never miss deadlines, their effective accuracy are simply equal to their baseline accuracies. The effective accuracy of the larger ResNet-50, -101, and -152 models reduces as the deadline constraints become tighter due to high deadline miss rates. In contrast, Model-Switching yields the highest effective accuracy for all deadline constraints — we note that this



(a) Model Switching (Target SLA deadline: 750 *ms*).

(b) Effective Accuracy.

(c) Empirical CDF of query latency.

Figure 4: Results on (a) 5-minute window of trace; (b) Effective accuracy; (c) Empirical CDF of query latency.

effective accuracy would be *unachievable* by the smaller models and only achievable by the larger models by introducing additional hardware resources.

Tail Latency To better understand how our Model-Switching adapts to load fluctuations, we plot the empirical CDF (Fig. 4c) of end-to-end latency observed by a client submitting requests in Fig. 4a. We compare the percentile latency of Model-Switching with baselines that serve requests using single model each. Several observations from the plot: (1) Small models, such as ResNet-18 and ResNet-34, guarantee that all queries finish within the deadline but have low baseline accuracies. (2) Large and more accurate models (e.g., ResNet-152 and ResNet-101) suffer long tail latency, resulting in several deadline misses. (3) Model-Switching, as it stands, seeks to combine the best of both worlds; i.e., meeting deadlines on the one hand, while serving requests using the most accurate models whenever possible.

CPU and RAM Usage: In all our experiments above, we allocate a total of 16 vCPUs (<R:4,T:4>) for active model containers and limit the CPU utilization for each of the inactive model containers to be less than 1%. 8 vCPUs are configured to handle the client's HTTP POST requests, and the remaining for other Clipper components. The Multi-Tenant ResNet models with a total 20 replicas occupy about 11.8% (15.1GB) of the total system RAM.

5 Related Work

There are an increasing number of studies on different aspects of MLaaS. The most relevant for our paper are those that introduce platforms and characterize their performance, and those that optimize QoS and resource allocation.

ML Inference Serving Framework A convention way to deploy ML service is to provision *containers*, or *servables* to host ML models. Examples include Clipper [10], Tensorflow Serving [31], and Rafiki [39]. These frameworks aim to minimize the cost of deployment, optimization of latency and throughput, and maintenance of DNN based MLaaS. Our work can be easily implemented on top of these existing frameworks with minimal modifications (indeed, we build our proposed solution within *Clipper*).

Auto-scaling Policies Auto-scaling policies are used to guarantee response time SLA while maximizing resource efficiency [3, 4, 18]. However, in the event of load spikes, these existing auto-scaling policies fail to capture the dynamics in time (since bringing up new hardware resources is time consuming) and increase resource usage. In this paper, we aim to solve this problem without scaling hardware resources but by exploiting model diversity while providing high QoS.

Model Accuracy Vs Performance Several works have attempted to optimize the model accuracy and performance. For example, static pruning (compression), quantization, and neural architecture search approaches [25, 36, 41, 43, 44] can generate a family of model versions that can be switched during run-time. Input-redundant techniques such as NoScope [27] and Focus [23] are primarily targeted at video queries (where a lot of redundancy exists in the transferred data between frames). In our case, we are targeting image (and presumably textual) queries from different users. In such scenarios, the opportunity to exploit input redundancy may be lower than that in video queries. Other input-dependent cascade methods [37, 40], also fit nicely with our model-switching framework wherein classifiers of different complexities could be switched in and out at run-time in response to the work load. Another related work [20] exploits model diversity by exposing latency/accuracy trade-offs to users, while we focus on automatically switching between models to optimize effective accuracy.

6 Conclusion

In this paper, we argue that for MLaaS, the *prediction accuracy* (or simply accuracy) of the model is also a critical metric but has not traditionally been encapsulated in SLAs. We call for *effective accuracy*, a new metric, that should be looked at when evaluating the performance of such systems. To achieve a better effective accuracy while serving the prediction requests, Model-Switching has been proposed to dynamically select the best model according to the load and the pre-characterized model performance. We evaluated our framework on a real MLaaS system.

7 Challenges and Discussion

Several challenges lie ahead of us before we can achieve our goal of an automatic and low cost Model-Switching controller for MLaaS.

Reducing Memory Overheads Containerization disallows any sharing of host machine resources. There is a trade-off between reducing cold start time and the memory resources. A model can be invoked quickly when it is already in memory and does not require a cold start. However, keeping all models in memory at all times is prohibitively expensive and does not scale well. Ideally, we want a method to provide illusion that all models are always warm, while spending resources as if they were always cold. Some work on this can be found in [5, 6].

Dynamic Replica and Thread Allocation Currently, we statically set the model replica and thread configuration before the deployment for simplicity. We are exploring more practical ways to implement this allocation online in real time.

Integration of Exiting Auto-scaling Framework In this work, we assume a fixed capacity C of computing resources (i.e. CPU cores) for backend serving. However, in practice, there may be a certain amount of resources available to scale up. In this setting, the problem of synergistically performing both model-switching and autoscaling remains open.

Offline Training-Free Model-Switching Controller Further, since we used the pre-characterized information about the P99 latency for each model at a fixed capacity. The model-switching controller needs to be retrained if autoscaling policy spawns or revokes some container replicas. As a future work, we are exploring the possibility of training a Reinforcement Learning agent to automatically learning these changes online.

Synergistic Optimization with Caching, Batching etc. Existing MLaaS frameworks enable performance optimizations through caching, adaptive batching [10, 19] etc. We need to first figure out what optimizations our model-switching is compatible with and also figure out a synergistic way to incorporate model switching into these techniques.

Extend to Multiple Types of Computing Resources Although CPUs are widely used for DNN inference in existing MLaaS platforms such as in Facebook [21] and Amazon [2], many specialized hardwares are designed for better DNN inference. Examples are GPU [38], FPGA [15], and Google's TPU [26, 42]. These heterogeneous hardware resources open new opportunities to optimize the latency, accuracy, power efficiency and resource efficiency, etc. with a holistic approach.

Acknowledgements

The authors would like to thank Jonathan Mace, as well as the anonymous reviewers for their time, suggestions, and valuable feedback.

References

[1] AI platform prediction. https://cloud.google.com/ai-platform/prediction/docs/overview/. Accessed: 2020-01-15.

[2] Amazon sagemaker ml instance types. https://aws.amazon.com/sagemaker/pricing/instance-types/. Accessed: 2019-09-30.

[3] Autoscaling in kubernetes. https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/. Accessed: 2020-01-15.

[4] Dynamic scaling for amazon ec2 auto scaling. https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html#as-how-scaling-policies-work. Accessed: 2020-01-15.

[5] Mikhail shilkov. cold starts in aws lambda. https://mikhail.io/serverless/coldstarts/aws/. Accessed: 2020-01-15.

[6] Mikhail shilkov. cold starts in azure functions. https://mikhail.io/serverless/coldstarts/azure/. Accessed: 2020-01-15.

[7] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016), pp. 265–283.

[8] Barrett, E., Howley, E., and Duggan, J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience 25*, 12 (2013), 1656–1674.

[9] Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., and Zhao, F. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI* (2008), vol. 8, pp. 337–350.

[10] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, Mar. 2017), USENIX Association, pp. 613–627.

[11] Curiel, M., and Pont, A. Workload generators for web-based systems: Characteristics, current status, and challenges. *IEEE Communications Surveys & Tutorials 20*, 2 (2018), 1526–1546.

[12] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[13] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), IEEE, pp. 248–255.

[14] FISCHER, W., AND MEIER-HELLSTERN, K. The markov-modulated poisson process (mmpp) cookbook. *Performance evaluation 18*, 2 (1993), 149–171.

[15] FOWERS, J., OVTCHAROV, K., PAPAMICHAEL, M., MASSENGILL, T., LIU, M., LO, D., ALKALAY, S., HASELMAN, M., ADAMS, L., GHANDI, M., ET AL. A configurable cloud-scale dnn processor for real-time ai. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (2018), IEEE, pp. 1–14.

[16] GANDHI, A., DUBE, P., KARVE, A., KOCHUT, A., AND ZHANG, L. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing (ICAC 14)* (2014), pp. 57–64.

[17] GANDHI, A., HARCHOL-BALTER, M., RAGHUNATHAN, R., AND KOZUCH, M. A. Distributed, robust auto-scaling policies for power management in compute intensive server farms. In *2011 Sixth Open Cirrus Summit* (2011), IEEE, pp. 1–5.

[18] GUJARATI, A., ELNIKETY, S., HE, Y., MCKINLEY, K. S., AND BRANDENBURG, B. B. Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (2017), pp. 109–120.

[19] GUPTA, U., HSIA, S., SARAPH, V., WANG, X., REAGEN, B., WEI, G.-Y., LEE, H.-H. S., BROOKS, D., AND WU, C.-J. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. *arXiv preprint arXiv:2001.02772* (2020).

[20] HALPERN, M., BOROUJERDIAN, B., MUMMERT, T., DUESTERWALD, E., AND REDDI, V. J. One size does not fit all: Quantifying and exposing the accuracy-latency trade-off in machine learning cloud service apis via tolerance tiers. *arXiv preprint arXiv:1906.11307* (2019).

[21] HAZELWOOD, K., BIRD, S., BROOKS, D., CHINTALA, S., DIRIL, U., DZHULGAKOV, D., FAWZY, M., JIA, B., JIA, Y., KALRO, A., ET AL. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2018), IEEE, pp. 620–629.

[22] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[23] HSIEH, K., ANANTHANARAYANAN, G., BODIK, P., VENKATARAMAN, S., BAHL, P., PHILIPOSE, M., GIBBONS, P. B., AND MUTLU, O. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (2018), pp. 269–286.

[24] JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia* (2014), pp. 675–678.

[25] JIANG, W., ZHANG, X., SHA, E. H.-M., YANG, L., ZHUGE, Q., SHI, Y., AND HU, J. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference* (2019), ACM, p. 5.

[26] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA, S., BODEN, N., BORCHERS, A., ET AL. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (2017), pp. 1–12.

[27] KANG, D., EMMONS, J., ABUZAID, F., BAILIS, P., AND ZAHARIA, M. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529* (2017).

[28] KRIZHEVSKY, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[29] LEMAY, M., LI, S., AND GUO, T. Perseus: Characterizing performance and cost of multi-tenant serving for cnn models. *arXiv preprint arXiv:1912.02322* (2019).

[30] MAO, M., LI, J., AND HUMPHREY, M. Cloud auto-scaling with deadline and budget constraints. In *11th IEEE/ACM International Conference on Grid Computing* (2010), IEEE, pp. 41–48.

[31] OLSTON, C., FIEDEL, N., GOROVOY, K., HARMSEN, J., LAO, L., LI, F., RAJASHEKHAR, V., RAMESH,

S., AND SOYKE, J. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[32] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., ET AL. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (2019), pp. 8024–8035.

[33] SAMANTA, A., SHRINIVASAN, S., KAUFMANN, A., AND MACE, J. No dnn left behind: Improving inference in the cloud with multi-tenancy. *arXiv preprint arXiv:1901.06887* (2019).

[34] SCHROEDER, B., WIERMAN, A., AND HARCHOL-BALTER, M. Open versus closed: A cautionary tale. USENIX.

[35] SRIRAMAN, A., AND WENISCH, T. F. μtune: Auto-tuned threading for {OLDI} microservices. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSD 18)* (2018), pp. 177–194.

[36] VENKATESH, G., NURVITADHI, E., AND MARR, D. Accelerating deep convolutional networks using low-precision and sparsity. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), IEEE, pp. 2861–2865.

[37] VIOLA, P., AND JONES, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition. CVPR 2001* (2001), vol. 1, IEEE, pp. I–I.

[38] WANG, L., WU, W., XU, Z., XIAO, J., AND YANG, Y. Blasx: A high performance level-3 blas library for heterogeneous multi-gpu computing. In *Proceedings of the 2016 International Conference on Supercomputing* (2016), pp. 1–11.

[39] WANG, W., GAO, J., ZHANG, M., WANG, S., CHEN, G., NG, T. K., OOI, B. C., SHAO, J., AND REYAD, M. Rafiki: machine learning as an analytics service system. *Proceedings of the VLDB Endowment 12*, 2 (2018), 128–140.

[40] WANG, X., LUO, Y., CRANKSHAW, D., TUMANOV, A., YU, F., AND GONZALEZ, J. E. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885* (2017).

[41] YANG, L., YAN, Z., LI, M., KWON, H., LAI, L., KRISHNA, T., CHANDRA, V., JIANG, W., AND SHI, Y. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. *Proceedings of the 57th Annual Design Automation Conference* (2020).

[42] ZHANG, J., RANGINENI, K., GHODSI, Z., AND GARG, S. Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *Proceedings of the 55th Annual Design Automation Conference* (2018).

[43] ZHANG, J. J., RAJ, P., ZARAR, S., AMBARDEKAR, A., AND GARG, S. Compact: On-chip compression of activations for low power systolic array based cnn acceleration. *ACM Trans. Embed. Comput. Syst. 18*, 5s (Oct. 2019).

[44] ZHANG, T., YE, S., ZHANG, K., TANG, J., WEN, W., FARDAD, M., AND WANG, Y. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 184–199.