Interference-Aware Scheduling for Inference Serving

Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis
{dmendo,faromero}@stanford.edu,{qianli,neeraja,christos}@cs.stanford.edu
Stanford University

Abstract

Machine learning inference applications have proliferated through diverse domains such as healthcare, security, and analytics. Recent work has proposed inference serving systems for improving the deployment and scalability of models. To improve resource utilization, multiple models can be co-located on the same backend machine. However, co-location can cause latency degradation due to interference and can subsequently violate latency requirements. Although interference-aware schedulers for general workloads have been introduced, they do not scale appropriately to heterogeneous inference serving systems where the number of co-location configurations grows exponentially with the number of models and machine types.

This paper proposes an interference-aware scheduler for heterogeneous inference serving systems, reducing the latency degradation from co-location interference. We characterize the challenges in predicting the impact of co-location interference on inference latency (e.g., varying latency degradation across machine types), and identify properties of models and hardware that should be considered during scheduling. We then propose a unified prediction model that estimates an inference model's latency degradation during co-location, and develop an interference-aware scheduler that leverages this predictor. Our preliminary results show that our interference-aware scheduler achieves 2× lower latency degradation than a commonly used least-loaded scheduler. We also discuss future research directions for interference-aware schedulers for inference serving systems.

CCS Concepts: • **Computer systems organization**; • **Computing methodologies** → *Machine learning*;

Keywords: machine learning, cloud computing, inference serving, interference-aware scheduling, heterogeneity

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroMLSys '21, April 26, 2021, Online, United Kingdom
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8298-4/21/04…$15.00
https://doi.org/10.1145/3437984.3458837

ACM Reference Format:
Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Interference-Aware Scheduling for Inference Serving. In *The 1st Workshop on Machine Learning and Systems (EuroMLSys '21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437984.3458837

1 Introduction

The prevalence of applications using machine learning inference is high and is expected to increase. Applications across various domains, such as healthcare [16], security [22], and analytics [17, 27, 33] are now relying on the inference process of the machine learning lifecycle (which accounts for over 90% of infrastructure cost on AWS [5]). Recently, several inference serving systems have been proposed to meet the growing demand for machine learning inference. These systems provide unified APIs for selecting and querying models across heterogeneous backends [10], automate resource scaling [32], configure model batch size based on application latency Service-Level Objectives (SLOs) [4, 15], and even select which model is most cost-effective for meeting application requirements [26].

To improve resource utilization and reduce cost, inference serving systems co-locate models on backend machines. To choose a backend machine for deploying new models, a common scheduling policy for existing systems is to choose the least-loaded worker (i.e., with lowest resource utilization). The least-loaded scheduler is heavily studied in the context of VM task scheduling and is a popular baseline in state-of-the-art task scheduling studies [11, 13, 29]. While the least-loaded scheduling policy has low overhead, model co-location can cause inference latency degradation due to interference, which can result in SLO violations. Further, the least-loaded policy is indifferent to varying inference latency across machine types, and thus least-loaded placement is likely to result in a sub-optimal inference latency. Interference-aware schedulers that leverage techniques such as collaborative filtering [11, 12, 25] and Bayesian Optimization [2, 24] have been previously proposed; however, they focus on long-running or static workloads, require extensive workload profiling, or cannot be used in heterogeneous execution environments.

To quantify the limitations of the least-loaded scheduler in inference serving, we performed 100 simulations in which a least-loaded scheduler places three models on two backend workers. We collected the latency degradation of each model

VM Type Options	Least-Loaded	Min
{V100, V100}	2.57	0.96
{V100, 32vCPU}	8.14	3.22
{V100, 16vCPU}	11.51	4.34

Table 1. Cumulative latency degradation (seconds) from 100 random inference workloads. V100 is an Nvidia GPU. The least-loaded scheduler is over 2× worse than an oracular minimum.

– the difference between the inference latency of serving a request during co-location and the inference latency of serving a request in isolation – and recorded the cumulative latency degradation over all simulations. Table 1 shows the cumulative latency degradation using the least-loaded scheduler compared to an oracular minimum that places models according to the lowest possible co-location latency degradation. By not considering the co-location impact on a model's inference latency, the least-loaded scheduler is more than 2× worse than the oracular minimum. Furthermore, existing systems either *reactively* re-schedule models that are interfered [26], which is often too late to avoid SLO violations, or do not mitigate interference at all [9, 10, 15].

Ideally, an inference serving system should determine what models can be safely co-located together on a hardware platform when deploying them to avoid SLO violations. However, this would require the system to assess how each model's latency is affected by being co-located with one or more models across different machine types. Furthermore, having to measure how co-location affects each new model is infeasible, especially as the number of models deployed continues to grow [26]. While there have been several prior frameworks and schedulers, such as Quasar [12], Paragon [11], Mage [25], Clite [24], and CherryPick [2] that have accounted for the interference effects of co-locating workloads on cluster machines, they are not suitable for inference serving systems for the following reasons. First, they focus on batch and long-running workloads where inaccurate profiling can be corrected as the application runs. Inference models typically serve requests on the order of milliseconds, and thus cannot be live migrated without incurring significant latency degradation. Second, they require substantial amounts of co-location configuration exploration and profiling. This makes it difficult to explore or profile an exponentially large number of co-located models registered for use by the inference serving system. Third, existing frameworks fix themselves to homogeneous systems and can only be applied to CPU only systems or GPU only systems. Thus, an inference serving system's scheduler should utilize a data-driven predictor that learns how co-locating model architectures on different machine types affects their latency.

In this paper, we discuss the challenges of co-locating inference models in an inference serving system, where models are deployed and scaled on heterogeneous machine types.

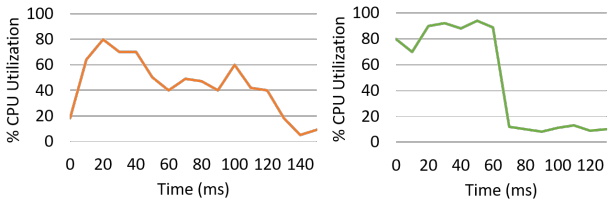


Figure 1. Utilization versus time (ms) for DenseNet201 (left) and VGG16 (right). Each curve shows the fluctuation in hardware utilization through inference execution, and illustrates the challenge in predicting co-location inference latency.

We propose an interference-aware scheduler that proactively considers the impact of co-location interference on latency to minimize latency degradation and reduce SLO violations. When deploying models to serve queries, the interference-aware scheduler predicts the latency degradation of co-location configurations, and selects the backend worker that minimizes the total predicted latency degradation from interference. To address the exponentially large number of possible co-location configurations, we propose a prediction model that is capable of exploiting the similarity of co-location configurations across inference model attributes and machine types to maximize prediction accuracy. Our initial evaluation on 14 models across 7 machine types on Google Cloud Platform [14] shows that our interference-aware scheduler reduces latency degradation by 2.6× compared to the least-loaded scheduler approach used by existing systems. We also discuss open research directions for scheduling in inference serving systems.

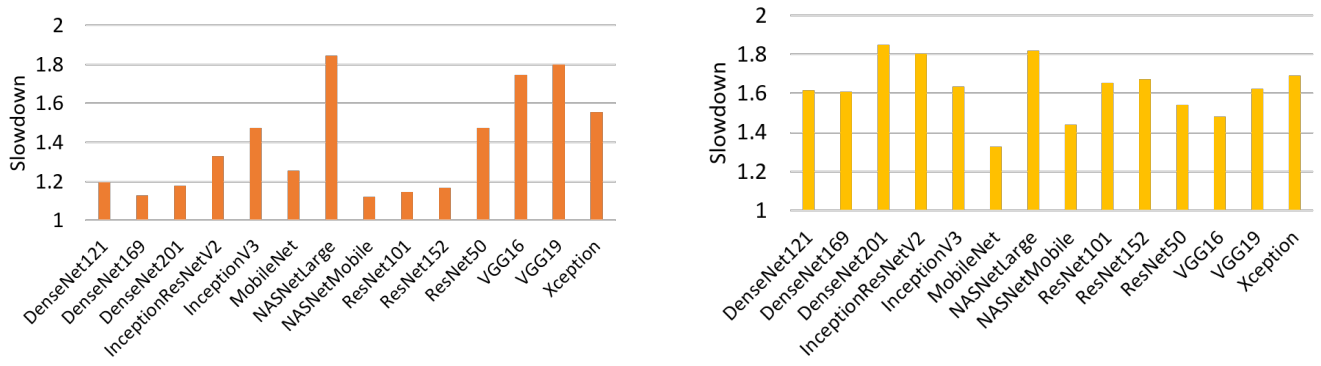
2 Motivation

In this section, we describe the challenges of developing an interference-aware scheduler for inference serving systems, and discuss why existing solutions are insufficient.

2.1 Challenges

Variations in hardware utilization. While a model serves an inference request, its hardware utilization can fluctuate throughout its execution. For instance, Figure 1 shows the utilization for a single request served by DenseNet201 and VGG16 on a 16vCPU VM. In both cases, the earlier phases of the execution exhibit higher utilization than the concluding layers since both models employ parallel convolution operations in the beginning of the inference execution followed by sequential matrix multiply operations in the concluding fully connected layers. This illustrates the complexity of the interaction between models and the hardware, and highlights the challenge of predicting how co-location will affect inference latency.

Latency variations from co-locating models across different machine types. As observed by INFaaS [26], the



(a) 24vCPU VM

(b) V100 GPU

Figure 2. Slowdown of DenseNet201 during co-location against isolation on a 24vCPU VM (left) and V100 GPU (right). Slowdown is the ratio of inference latency during co-location versus run in isolation. This result exemplifies the high variance of latency degradation across co-located models and machine types.

amount of latency degradation due to co-location interference varies based on the model and its underlying machine type. For example, Figure 2 shows the slowdown for DenseNet201 when co-located with different model architectures on a 24vCPU VM and on an NVIDIA V100 GPU. Slowdown is the ratio of inference latency during co-location versus isolation. We observe that the slowdown differs across models: co-location with models such as NASNetLarge can be as high as 1.8× while others such as ResNet101 are less than 1.2×. This variation is caused by the architecture and resource requirements of different models. For instance, the architecture of NASNetLarge is highly parallel and contains more weights compared to the other 13 models, and thus highly demands compute and memory resources. Thus, NASNetLarge causes higher resource contention during co-location. Furthermore, the slowdown varies on different machine types. For instance, DenseNet201 caused high slowdown on V100 but low slowdown on a 24vCPU VM. However, for a few models, such as NASNetLarge, the slowdown is similar on both machine types.

More than two models may co-locate on the same machine. We refer to the number of co-located inference models on a single machine as the *degree of co-location* (e.g., two in Figure 2). We have also observed the pattern between the co-location degree and the latency degradation varies based on the underlying hardware and models, which makes analyzing latency degradation more difficult. This further motivates the need for an efficient and accurate latency degradation predictor as the degree of co-location varies.

2.2 Related Work and Existing Frameworks

The fluctuating hardware utilization, and the latency degradation caused by co-location across different models and machine types makes predicting latency degradation difficult. Existing work in interference-aware scheduling does not address these complexities. Bayesian Optimization approaches such as CherryPick [2] and Clite [24] automatically find the best static resource partitioning for general workloads on CPUs. However, static resource partitioning does not fit well for inference serving, since resource demand constantly fluctuates as inference queries are served. Inference execution is likely to either, on average, under-utilize or over-utilize a static resource partitioning due to fluctuation of utilization. Quasar [12], Paragon [11], and Mage [25] use a collaborative filtering-based technique for interference-aware scheduling on CPU servers. These frameworks focus on periodic and long-running workloads. Thus, their co-location configuration can be adjusted if needed. In inference serving, models are frequently loaded and unloaded based on the input query rate and application requirements. This makes it difficult for co-location decisions to be revisited and adjusted.

The frameworks discussed above require profiling co-location configurations on every worker machine type in the system. Given that inference serving systems often have thousands of inference models, and the number of co-location configurations is exponentially large, the profiling requirement of these systems is likely computationally infeasible.

Xu *et al.* proposed an interference-aware framework for inference workloads on GPU servers [31]. However, this framework is limited to models running on GPUs, and only considers pairs of models being co-located.

3 Interference-Aware Scheduling

Our proposed interference-aware scheduler predicts the latency degradation of co-locating models, and decides when it is safe to do so without violating application latency requirements. The interference-aware scheduler uses a predictor that estimates the latency degradation of a model co-located with one or more of the models that are running on each backend worker machine. Given the complexities of fluctuating hardware utilization and varying co-locating

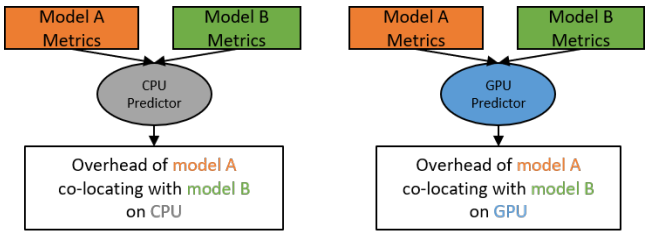


Figure 3. Diagram of separate predictors on each machine type.

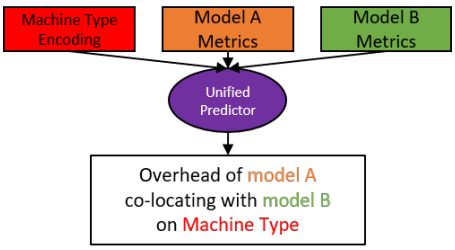


Figure 4. Diagram of a unified predictor across machine types.

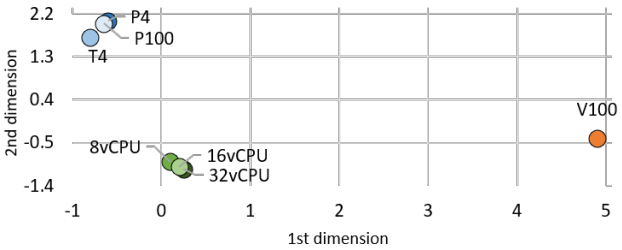


Figure 5. Learned 2D encodings of machine types. 8/16/32vCPU VMs have similar encodings, while T4/P4/P100 GPU VMs have similar encodings.

behaviors across machine types and models, there is no clear mapping between metrics of the inference models and the co-location latency degradation. Thus, we propose using a machine learning model that learns the mapping between the characteristics of inference models and the co-location latency degradation.

3.1 Separate Predictors

To predict latency degradation, one approach to is to train multiple predictors for different co-location degrees and machine types (shown in Figure 3). Each predictor is dedicated to predicting the latency degradation on a particular machine type and captures a different architecture's behavior. The separation of tasks allows for each predictor to be developed independently from each other, which makes this approach amenable to highly varying co-location latency degradation across machine types. However, the approach also has a few disadvantages. Given a maximum co-location degree C and number of unique machines types M, the approach requires training and maintaining $C \times M$ latency degradation predictors. Further, the separate predictors cannot exploit similarity across co-location configurations. As we will show, the co-location behavior of certain models are similar for some machine types, such as CPUs with different number of cores or GPUs. Given the large space of possible model co-location configurations, exploiting similarity across them for predictions is important to train an accurate predictor for minimizing the amount of profiling needed.

3.2 Unified Predictor

To address the inability of separate predictors to exploit similarity across machines types, we propose a unified predictor that can identify similarities across model co-location configurations (i.e., similarity of co-location behavior between models and machine types). Figure 4 illustrates the unified prediction approach. The unified predictor's inputs are (a) each inference model's metrics, and (b) a machine type encoding.

Advantages over separate predictors. The unified predictor has several advantages over separate predictors. First, the unified predictor leverages similarity across machine types for accurate prediction. For instance, models co-located on a 8vCPU VM may exhibit similar co-location behavior running on a 32vCPU VM; a unified predictor trained on both 8vCPU VM and 32vCPU VM data can exploit this similarity to improve accuracy. Second, the unified predictor is capable of using profiled model metrics on one machine type to predict on another machine type. This enables developing a latency degradation prediction approach for machine types that do not have tools for profiling, such as TPUs [18]. Third, only one unified predictor needs to be trained as opposed to CM models in the separate prediction approach.

To show the advantage of the unified approach against the separate approach, we implemented the separate predictors and unified predictor using a random forest regressor. Each prediction approach was trained on an identical 70-30 train-test split containing co-location latency degradation data from 8/16/32vCPU VMs and V100/P100/T4/P4 GPUs. The unified predictor approach achieved a 12.5% lower Mean Absolute Error (MAE) than the separate predictors method.

Machine type encodings. The unified approach for predicting model co-location latency degradation requires a machine type encoding as input to specify the machine type for the co-location configuration. A simple encoding scheme is a one-hot encoding for each machine type (used in the comparison to the separate predictor approach). However, this approach does not represent the similarity of co-location behavior across machine types well. For instance, intuitively, an 8vCPU VM may have similar co-location behavior as a 16vCPU VM. An encoding that captures similarity is likely to improve the prediction accuracy.

To capture similarity across machine types, we propose to learn the embedding, as is done with natural language models. As a prototype, we implemented a neural network

that learns machine type encoding to predict the similarity between machine types (based on the pearson correlation coefficient [7]). Figure 5 shows the results of a learned 2D encoding. As we expect, the CPU machine type encodings are closely clustered, and are thus similar. We see a similar clustering with GPU machine types. These encodings aid the latency degradation predictor in identifying similarities of co-location behavior across machine types and improve prediction accuracy, especially in data-limited settings in which not much data for a given machine type is available.

To test whether the encodings actually improve prediction accuracy in situations where profiling on every machine type is infeasible, we experimented with a situation in which a latency degradation prediction model is trained without 16vCPU VM data. Table 2 shows the MAE on the test set of a random forest regression model that is trained without any 16vCPU VM data. We note that using the learned machine type encodings reduced MAE by 32% compared to the one-hot encoding approach.

Approach	MAE on all machines	MAE on 16vCPU VM
One-hot	0.0305	0.1069
Learned	0.0206	0.0432

Table 2. Mean Absolute Error (MAE) on test split for latency degradation predictor trained without 16vCPU VM data. By learning the machine type encoding, we reduce the MAE.

Prediction features. To predict the latency degradation for co-located models, we need to provide the unified predictor with input features that relate to co-location behavior of the model on each machine type. We use hardware metrics of models running in isolation on each machine type. These metrics are outlined in Table 3 for each machine type.

Machine Type	Model Metrics
CPU	Util., DRAM usage
GPU	Util., Global buffer usage, PCIE read BW, PCIE write BW

Table 3. Hardware metrics used as features for prediction on CPU and GPU machine types.

When profiling hardware metrics is not possible, our insight is that we can instead profile metrics on a subset of all machine types in a heterogeneous system. The profiled metrics across machine types may be capturing similar information (e.g., an inference model in isolation that exhibits high CPU utilization may also exhibit high GPU utilization). If the metrics capture the similar information, then we expect that profiling on a subset of machine types would result in low loss in prediction accuracy. For instance, we implemented the unified predictor using a random forest regressor trained on a 70-30 train-test split trained on co-location latency degradation data on a 8/16/32vCPU VM, and V100/P100/T4/P4 GPUs. First, we trained the model with the input features as the hardware profiled metrics from both CPU and the V100. Next, we trained another model with the input features as the hardware profiled metrics from only the V100 and excluded the CPU metrics. In Table 4, we observe that excluding CPU metrics in the input features for the model increases the MAE on the test set by only 2%. These results suggest that collecting hardware profiled metrics on a subset of all machine types in the heterogeneous system is sufficient for training an accurate latency degradation predictor.

Features	CPU & GPU Metrics	GPU Metrics	Architectural
MAE	0.0146	0.0149	0.0164

Table 4. Mean Absolute Error (MAE) on test split (using 70-30 train-test split) of the unified predictor trained with both CPU and GPU metrics, with only GPU metrics, and with inference model architecture features. Profiling on a subset of all machine types incurs a negligible accuracy loss.

Model architecture features can also be directly used as a set of features for the predictor. As we noted in Section 2.1, the architecture's layout (e.g., number of input layers, hidden layers, and output layers) has an impact on the utilization, and the inference latency. Thus, we also evaluated the unified predictor trained with architectural features of the inference models. The architectural features are inputted as a list containing an entry for each layer in the model. Each entry consists of the layer's operation type (e.g., dense versus convolutional) and the number of weights in the layer. As we see in Table 4, the unified predictor with the architectural features perform slightly (9%) worse than using just hardware metrics. Further investigation into how to best use architectural features with the unified predictor is part of our future work (see Section 5). However, we suspect the difference in MAE is caused by the high dimensionality of the architectural approach when comparison to the hardware-profiled metrics approach, since inference models have tens to hundreds (or even thousands) of layers.

3.3 Proactive Scheduling

With a proactive scheduler, decisions are quantitatively evaluated a-priori, and actions are chosen to optimize the evaluation metric. In inference serving systems, a latency degradation predictor can be used to evaluate model placement for each query.

We now describe how the co-location latency degradation predictor is used for interference-aware scheduling. Let τ_i^j denote the predicted inference latency for inference model i running on worker machine j. The interference-aware scheduler selects an inference model i and places the model onto

a worker machine j that minimizes:

$$C_i^j = \tau_i^j + \sum_{y \in \texttt{Workers}} \sum_{x \in \texttt{Running}(y)} \tau_x^y$$

For model deployment of a given inference model v, the scheduler places the model v on machine u which has the lowest C_v^u. For each inference model i and worker machine j, C_i^j is pre-computed and stored so that predictions occur off of the critical path of serving queries and deploying models. When a new model is loaded or unloaded on a worker machine z, for each inference model i, C_i^z is concurrently recomputed given the new state of the worker machine. This update to C_i^z is linear in the number of registered inference models, and is embarrassingly parallel across each inference model (i.e., C_i^z can be computed in parallel).

Scheduler scalability. Since latency degradation predictions for each inference model are computed prior to query serving, determining the model placement involves finding the minimum element of a list whose size is equal to the number of worker machines. Thus, the worst case time complexity to identify the model placement is linear according to the number of worker machines. In addition, finding the minimum element in a list is parallelizable, which in the best case achieves logarithmic time complexity according to the number of worker machines. The scheduler can also trade off accuracy for speed by leveraging the power-of-two choice technique, which exhibits constant time complexity irrespective of the number of worker machines [23]. Therefore, we expect the scheduling overhead to remain low even as we scale from tens to hundreds of worker machines. We also note that the optimality of scheduling decisions is based on the accuracy of the latency degradation predictor. Since the latency degradation prediction performance is independent of the number of worker machines, we do not expect the optimality of scheduling decisions to be impacted when scaling the number of worker machines.

4 Preliminary Results

Implementation. We implemented the interference-aware scheduler as a simulation. The simulation framework is the same used to obtain the results in Table 1. We executed 100 simulations in which the interference-aware scheduler needed to choose and place three models on two worker machines. The framework emulates an inference application with three independent query sources with fixed accuracy constraints at maximum query rate and two worker machines. We collected the inference latency profiles of 14 pre-trained Keras [20] image recognition models that include DenseNet, NASNet, VGG, and ResNet architectures. We used three machine types on Google Cloud Platform [14]: 16vCPU VM (`n1-standard-16`), 32vCPU VM (`n1-standard-32`) and V100 GPU (`n1-standard-12` with V100).

Preliminary results. We now show that an interference-aware scheduler can improve latency degradation compared to a least-loaded scheduler. Table 5 shows the total cumulative latency degradation of the same random workload executed by the interference-aware scheduler, the least-loaded scheduler, and the minimum possible latency degradation on a given system of worker machines (i.e., the oracular scheduler). The interference-aware scheduler outperforms the least-loaded scheduler and achieves more than 2× less cumulative latency degradation. Further, the interference-aware scheduler with each system of worker machines close to the minimum possible latency degradation. The interference-aware scheduler exhibits an advantage over the least-loaded approach since it is able to reduce co-location latency degradation, which is important for avoiding SLO violations.

In future work, we plan to evaluate the interference-aware scheduler on an inference serving system with tens to hundreds of diverse worker machines. However, since the latency degradation prediction accuracy is independent of the number of workers, we expect our preliminary results to continue outperforming the baselines as the number of worker machines grows.

VM Type Options	Least-Loaded	Inter-Aware	Min
{V100, V100}	2.57	1.51	0.96
{V100, 32vCPU}	8.14	4.11	3.22
{V100, 16vCPU}	11.51	4.49	4.34

Table 5. Cumulative latency degradation (in seconds) from 100 random inference workloads. Interference-aware scheduler nearly matches the oracular minimum, and has latency degradation more than 2× lower than least-loaded scheduler.

5 Future Work and Research Directions

Active learning for predictor training. Developing a co-location latency degradation predictor requires collecting training data by profiling the different co-location configurations. The exponentially-large model co-location configuration space makes it infeasible to profile all possible configurations. Thus, selecting which configurations to profile is important to both minimize training time and maximize accuracy. However, selecting which configurations to profile is non-trivial. Figure 6 shows the expected MAE of the unified predictor on a 16vCPU VM given 14 unique CNNs against the size of the training set when randomly sampling training examples over 2000 simulations. The red curve shows the average performance of randomly sampling training data, while the blue curve shows the minimum MAE observed over 2000 simulations. The blue curve shows that the predictor benefits from strategically selecting which samples to profile and train with.

To strategically select which samples to use for training the predictor, we plan to explore active learning algorithms.

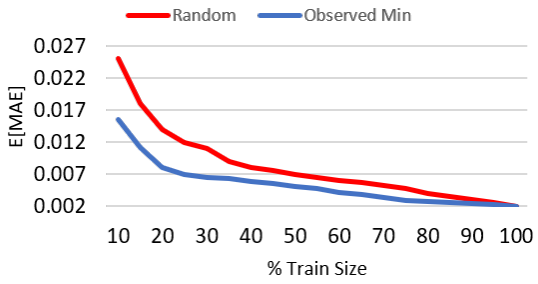


Figure 6. Expected Mean Absolute Error (MAE) when training the co-location latency degradation predictor. `Random` randomly samples training inputs, while `Observed Min.` is the observed minimum MAE.

These algorithms identify a subset of the space that can be profiled in a tractable amount of time to maximizes prediction accuracy [6]. Active learning methods have shown to perform better than random sampling [28], which can help in developing accurate latency degradation predictors without having to exhaustively profile all co-location configurations.

Few-shot transfer learning. Inference serving environments continue to become more complex as the number of machine types, models, hardware platforms, and compilers [1, 8] or graph optimizers [3] continue to increase. These factors make it difficult to maintain accurate latency degradation predictors, which will need to be frequently retrained, each time with a growing number of data points. We plan to explore whether few-shot transfer learning can be leveraged for quickly adapting our proposed predictor to support new hardware platforms and models (including optimized versions of models).

Extensibility to co-location prediction for power. When deploying models, especially at scale, providers typically want to minimize power consumption. We would like to explore whether our unified predictor can be extended to also predicting the power dissipation of inference models during co-location. Given some characteristics of co-located inference model and a machine type encoding, it may be possible to train the predictor to estimate the power dissipation of co-located inference models on a worker machine. The challenge lies in determining whether our proposed machine type encodings and model metrics are sufficient for predicting power consumption, or whether we would need to extract further metrics from the underlying hardware.

Clear-box architectural features for predicting co-location latency. Our initial results using clear-box architectural features (e.g., output dimension of each layer and layer types) show that the unified predictor exhibits slightly higher MAE than if trained on the hardware-profiled metrics. However, there are other architectural aspects of the inference models which may be indicative of co-location behavior, such as layer connectivity, that we plan to explore in future work. For instance, prior studies have shown success in predicting the inference latency during isolation on TPUs with architectural features of the inference models [19].

Reinforcement learning for scheduling. Another research direction is to investigate Reinforcement Learning (RL) approaches for scheduling. RL planning methods are guided by regret, which is a quantity that measures how undesirable the effect of an action is at a certain state. An RL planning algorithm chooses actions that minimizes estimates of future regret. Interference-aware scheduling can be framed as a planning problem, and its regret can be defined as the cumulative inference latency overhead. However, accurately estimating future regret is challenging since the future query workload is unpredictable. Decima [21] is an example of a scheduler that uses RL for Spark jobs. However, Decima is likely not suitable for inference serving as Spark jobs tend to be long running and do not run on heterogeneous hardware platforms.

The interference-aware scheduling problem can also be framed as a contextual multi-armed bandit. The classic multi-armed bandit problem is introduced as a slot machine problem where there are n arms each associated with a probability of success [30]. Systems such as Clipper [10] use the multi-armed bandit approach for model selection to provide more accurate and robust predictions. In the context of interference-aware scheduling, the arms represent the worker machines of the system where the scheduler must choose which worker machine to place an inference model. This approach frames interference-aware scheduling as an end-to-end prediction task, which is likely to exhibit fast model deployment. However, end-to-end prediction tasks often require orders of magnitude more training data.

Accurately modeling a future workload may improve feasibility of RL planning and multi-armed bandit methods for interference-aware scheduling.

6 Conclusion

We proposed a interference-aware scheduler for inference serving systems. We observed unique characteristics of inference workloads in heterogeneous environments, and proposed an strategy to predict the latency degradation due to co-locating inference models. We showed that our prediction approach enables exploitation of similarity across inference model and machine types to achieve more accurate predictions with less training data. We implemented a simulation framework to prototype our scheduler, and showed the interference-aware approach achieved up to 2.6× lower total latency degradation compared to a least-loaded scheduler. Finally, we discussed future research directions that will be key to designing and implementing efficient and accurate interference-aware schedulers for inference serving systems.

References

[1] 2018. *NVIDIA TensorRT: Programmable Inference Accelerator.* https://developer.nvidia.com/tensorrt.

[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 469–482. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[3] AWS [n.d.]. AWS Neuron. https://github.com/aws/aws-neuron-sdk.

[4] AWS 2018. AWS Inferentia. https://aws.amazon.com/machine-learning/inferentia/.

[5] AWS 2019. Deliver high performance ML inference with AWS Inferentia. https://d1.awsstatic.com/events/reinvent/2019/REPEAT_1_Deliver_high_performance_ML_inference_with_AWS_Inferentia_CMP324-R1.pdf.

[6] Maria-Florina Balcan and Ruth Urner. 2016. *Active Learning – Modern Learning Theory.* Springer New York, New York, NY, 8–13. https://doi.org/10.1007/978-1-4939-2864-4_769

[7] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 37–40.

[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[9] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 477–491.

[10] Daniel Crankshaw, Xin Wang, Giulio 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 2017, Boston, MA, USA, March 27-29, 2017*. 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[11] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/2451116.2451125

[12] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.* 49, 4 (Feb. 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[13] D. Chitra Devi and V. Rhymend Uthariaraj. 2016. Load Balancing in Cloud Computing Environment Using Improved Weighted Round Robin Algorithm for Nonpreemptive Dependent Tasks. *The Scientific World Journal* 2016 (03 Feb 2016), 3896065. https://doi.org/10.1155/2016/3896065

[14] Google [n.d.]. Google Cloud Platform. https://cloud.google.com/compute.

[15] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-scale Neural Recommendation Inference. arXiv:2001.02772 [cs.DC]

[16] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qia ng Dong, Haipeng Shen, and Yongjun Wang. 2017. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology* 2, 4 (2017), 230–243. https://doi.org/10.1136/svn-2017-000101 arXiv:https://svn.bmj.com/content/2/4/230.full.pdf

[17] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 253–266. https://doi.org/10.1145/3230543.3230574

[18] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[19] Samuel J. Kaufman, Phitchaya Mangpo Phothilimthana, Yanqi Zhou, and Mike Burrows. 2020. A Learned Performance Model for the Tensor Processing Unit. arXiv:2008.01040 [cs.PF]

[20] Keras [n.d.]. Keras. https://github.com/fchollet/keras.

[21] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 270–288. https://doi.org/10.1145/3341302.3342080

[22] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. 2020. Shredder: Learning Noise Distributions to Protect Inference Privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/3373376.3378522

[23] M. Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104. https://doi.org/10.1109/71.963420

[24] T. Patel and D. Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 193–206. https://doi.org/10.1109/HPCA47549.2020.00025

[25] Francisco Romero and Christina Delimitrou. 2018. Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques* (Limassol, Cyprus) *(PACT '18)*. Association for Computing Machinery, New York, NY, USA, Article 19, 13 pages. https://doi.org/10.1145/3243176.3243183

[26] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2019. INFaaS: Managed & Model-less Inference Serving. *CoRR* abs/1905.13348 (2019). arXiv:1905.13348 http://arxiv.org/abs/1905.13348

[27] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. arXiv:2102.01887 [cs.DC]

[28] Nicholas Roy and Andrew McCallum. 2001. Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 441–448.

[29] Subhadra Shaw and A. Singh. 2014. A survey on scheduling and load balancing techniques in cloud computing environment. 87–95. https://doi.org/10.1109/ICCCT.2014.7001474

[30] Joannès Vermorel and Mehryar Mohri. 2005. Multi-Armed Bandit Algorithms and Empirical Evaluation. In *Proceedings of the 16th European Conference on Machine Learning* (Porto, Portugal) *(ECML'05)*. Springer-Verlag, Berlin, Heidelberg, 437–448. https://doi.org/10.1007/11564096_42

[31] Xin Xu, Na Zhang, Michael Cui, Michael He, and Ridhi Surana. 2019. Characterization and Prediction of Performance Interference on Mediated Passthrough GPUs for Interference-aware Scheduler. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotcloud19/presentation/xu-xin

[32] 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)*. USENIX Association, Renton, WA, 1049–1062. https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[33] 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)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang