Predicting the Performance of DNNs to Support Efficient Resource Allocation

Sarah Shah
Electrical and Software Engineering
University of Calgary
Calgary, Canada
sarah.shah1@ucalgary.ca

Yasaman Amannejad
Mathematics and Computing
Mount Royal University
Calgary, Canada
yamannejad@mtroyal.ca

Diwakar Krishnamurthy
Electrical and Software Engineering
University of Calgary
Calgary, Canada
dkrishna@ucalgary.ca

***Abstract*—Numerous organizations are adopting sophisticated Machine Learning (ML) algorithms for their operations. To ensure the optimal performance of ML systems, organizations require insights into the response time of such systems under realistic user workloads. However, despite the widespread adoption of ML models, research on *predicting* the response time of a system serving an ML model under varying resources and user workloads is limited. In this paper, we address this gap by proposing a modeling approach to predict response times of multiple well-known Deep Neural Networks (DNNs) under simultaneously varying resource settings and user workloads. We join a classifier and a regressor to identify the optimal resource setting for meeting a DNN's response time target, and to predict the response time under the allocated resource setting. Our technique enables performance modeling without the need to collect extensive data during system operation, thus empowering pre-deployment predictions. The results demonstrate that our approach can generalize to unseen resource and workload scenarios, guaranteeing accurate predictions of compliance with response time targets 98.05% of the time and offering response time predictions with a mean prediction error of 9.10%.**

***Index Terms*—Machine Learning, Performance Modeling, Inference Time Prediction, Deep Neural Networks, TensorFlow Serving.**

I. Introduction

Organizations are increasingly leveraging ML for their operations [1]. DNNs have particularly emerged as the core to support complex tasks such as image recognition and Natural Language Processing (NLP). ML serving systems support real-time ML predictions by hosting ML models on servers and providing an API for users to use models. TensorFlow Serving [2], developed by Google, is a popular serving platform. It offers a user-friendly interface for ML developers to serve and update models in real time, with the ability to switch between different model versions seamlessly. It can also be containerized using Docker [3].

Often, DNNs are deployed for making time-sensitive decisions, e.g., object detection in self-driving vehicles and online fraud detection, thus requiring systems hosting ML models to offer inference results within desired response time targets. However, this can become a challenge due to varying inference workloads. Additionally, the serving platform's operation should be cost-effective. Thus, it is crucial to allocate resources *efficiently*, for instance by using container resource allocation mechanisms [4], [5], to obtain a desired response time without over-provisioning. However, limited research has been conducted on predicting the performance of ML serving systems as a function of resources and workloads.

Currently, there is limited research for predictive insights on how to configure a DNN serving system to meet response time targets. ML research has made progress in optimizing response time through techniques like model pruning, quantization, and modifying model structure [6]. These approaches reduce the model's size, complexity, and inference time. However, in many cases, pre-built models are used without the option to rebuild them for improved performance. Many existing studies primarily focus on *characterizing* response times of DNNs but lack performance modeling for future serving scenarios. Current research on performance prediction of ML models [7]–[9] is limited in scope or has not yielded promising results.

We distill several requirements for an effective performance prediction technique. Given the high likelihood of DNN serving systems encountering diverse request rates and resource configurations, it is crucial that the performance prediction technique demonstrate *generalizability* by effectively predicting for unseen situations. Moreover, organizations often have multiple DNNs hosted on their servers to perform different tasks. Thus, having a single performance prediction model for all deployed DNNs being served at an organization can be advantageous. It is also beneficial to not rely on collecting performance metrics during request inference so that predictions can be offered in advance to help proactively allocate the right amount of resources to meet a response time target. This also avoids any overhead introduced by metric collection during inference. Lastly, apart from guaranteeing compliance with response time targets, a performance prediction technique must predict system response time to facilitate tasks such as anomaly detection and task scheduling. None of the existing techniques for DNN performance prediction embody all of these requirements.

Focusing on the above requirements, we aim to fill the existing gaps in the DNN performance prediction literature by answering the following Research Questions (**RQs**):

- **RQ1**: How do DNN response times vary based on workloads and resources available to the serving platform?

- **RQ2**: Which modeling techniques can help develop a performance model that can facilitate compliance to response time targets *and* predict response times in a generalizable manner for performance monitoring and regulation?
- **RQ3**: Can a single unified performance model effectively predict the performance of multiple DNN types?

We answer the above questions by making the following contributions in this paper:

- We scrutinize the impact of resource and workload factors on the response time of different DNNs (**RQ1**). We note that DNN serving systems exhibit queuing behavior which varies across different DNNs and workloads.
- We develop a novel hybrid ML-based performance prediction approach for multiple well-known DNNs that offers accurate performance insights generalizable to unseen scenarios (**RQ2**). Our approach comprises two steps: a classification of whether a workload under a given resource configuration will meet the DNN's response time target; and if so, a prediction for the response time for that workload under the prescribed resource configuration. Our classifier achieves an average accuracy of 98.05% on 6 previously unseen test sets. This classifier is then concatenated by a regressor that predicts response times with an average prediction error of 9.10%, thus enabling performance decisions and regulation.
- Results indicate that our unified modeling technique is effective for diverse DNNs (**RQ3**) with varying tasks, architectures, and resource requirements.

The rest of the paper is organized as follows. Section II offers an overview of the existing relevant literature. Section III describes the proposed process for developing the performance model. Section IV describes the experimental setup and evaluation metrics. Section V describes the results of our analysis in detail. Lastly, Section VI summarizes our contributions and possible future extensions.

II. Related Work

Most research efforts in ML inference area are focused on inference optimization [6], [10]–[13] and characterization [14]–[16]. Previous studies [7], [9], [17] have reported that the inference time of ML models can vary significantly according to the allocated resources and workloads. This makes the building of a model to predict the inference time of ML models to be a complex task.

We first review literature on ML serving platforms. Gujarati et al. [18] present Clockwork, an ML serving system with predictable performance. Clockwork ensures only a single request is executed at a time because simultaneous executions can make the scheduler unpredictable. Thus, it limits the workers to execute only one request at a time in order to maintain the predictability of the framework, at the cost of a small decrease in throughput. Zhang et al. [19] propose MArk that aims to minimize serving costs while respecting SLA limits by batching inference requests and serving them on hardware accelerators for faster inference. The prediction system predicts anticipated workloads. The authors point out that they do not intend to offer an optimal prediction model, but aim to demonstrate that workload prediction can help save costs. In contrast, our work is focused on predicting the inference time of ML models.

We next review the performance prediction of ML inference. Li et al. [9] study DNN inference under SLA and cost constraints and propose an automated deployment framework by combining Bayesian Optimization and Deep Reinforcement Learning. They report a decrease in inference time compared to Google's basic device placement. Their approach works only for cloud-based ML deployment. Moreover, they only model CPU and GPU units as resources, excluding other resources such as memory. Cai et al. [8] use a repository of model variants and their performance measurements to automatically select minimal GPU resources required to meet an SLA during ML inference under a given workload. In their evaluations, they show that they meet SLAs in 94.5% of instances with a maximum prediction error of 17% for inference time predictions. Their study is catered specifically to GPU deployment and focuses only on image classification models. Fu et al. [7] study the possibility of developing a unified predictive solution for the latency of 13 non-ML and ML applications. They employ 6 ML models as black boxes for the performance prediction of the 13 applications under varying input sizes, amounts of worker nodes, and types of worker nodes. The paper reports that accurately predicting the behavior across diverse applications was a challenge with prediction errors reaching as high as 128.6%. Consequently, the paper concludes that developing a single performance prediction model for multiple applications is a challenge.

From the discussions above, we note that there is a gap in the literature for generalized accurate prediction models for DNN inference. We need these performance models to make the necessary allocation of resources for inference time targets to be met under diverse request rates. While the studies mentioned above do attempt to provide predictive models for DNNs, they are either unable to reach prediction errors low enough to be pragmatically used, or are centered around a particular type of resource or ML task.

III. Methodology

We undertake a systematic investigation of the performance of different DNNs. We study their inference times with respect to 5 predetermined external parameters which are anticipated to influence DNNs' performance in real-world scenarios. These parameters are known prior to serving a DNN and before any inference requests are made. These include the DNN being served (*DNN Name*), the processors allocated to the container serving the DNN ($Processors$), the memory allotted to the container ($Memory$), the number of payloads in the inference request(s) (*Payload Instance*), and the rate at which those requests are sent ($Rate$). $Rate$ signifies the number of requests sent to the DNN server per second, hence dictating the number of concurrent requests handled by the DNN at a given time. *Payload Instance* refers to the data sent within a request for inference. $Rate$ and *Payload Instance* and describe the *workload* sent to the server, whereas $Memory$ and $Processors$ describe the *resources* used to serve that workload.

We aim to develop a performance model that represents the response time of DNNs by employing diverse ML algorithms. Specifically, we employ ML algorithms to capture the response times of ML models served on TensorFlow Serving. However, considering the limitations in accuracy found in previous approaches [7] that treat ML as a black-box for performance modeling, we adopt a systematic and incremental approach to develop our ML-based performance model, with the goal to predict performance such that response time targets may be met while guaranteeing effective performance monitoring and regulation.

Organizations typically require response time information to ensure compliance with stakeholder and user-specified deadlines. Therefore, our primary goal is to model the likelihood of meeting a predefined response time target instead of directly predicting the response time. By framing the problem as a classification task, we determine whether a specific workload will satisfy a predefined response time threshold given a particular resource configuration.

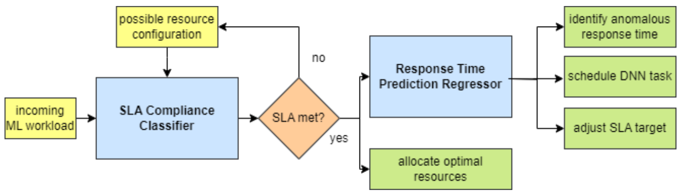

possible resource configuration
incoming ML workload
SLA Compliance Classifier
SLA met?
no
yes
Response Time Prediction Regressor
identify anomalous response time
schedule DNN task
adjust SLA target
allocate optimal resources

Fig. 1. Decision process using the proposed hybrid model

While predictions for compliance with response time targets could be enough to deploy a model on a serving platform with just the right amount of resources, accurate response time predictions are still crucial for monitoring and improving the system's performance. Therefore, we propose a hybrid modeling technique where we concatenate a classification-based *SLA compliance prediction* model with a regression-based *response time prediction* model, as depicted in Figure 1. The classifier determines whether a specific DNN will complete a request within the response time target specified by a predefined SLA, given a particular workload and resource configuration. If determined that the response time target will be met, the regression model predicts the expected response time for the DNN to handle the workload under the classifier-prescribed resource configuration. While the SLA-compliance prediction assists in appropriate resource allocation to ensure adherence to the response time targets, the response time predictions aid in making a variety of important run-time decisions, such as task scheduling to ensure SLA compliance for all tasks within the system, early identification, relocation, and cancellation of tasks with abnormally high response times, and response time target adjustments. Empirically, we noticed this two-pronged hybrid approach to be more accurate than directly predicting response times.

We train 8 different ML algorithms described in Section IV-C and fine-tune the algorithm that is most effective for modeling our data. Our objective is to create a performance model capable of providing predictions for resource-workload scenarios that have not previously encountered (**RQ2**). In real-world deployment environments, resource allocation is often determined by incoming workloads and the defined response time targets for each DNN inference. It is possible that the DNN has not previously experienced certain workloads as well as the resource settings used to serve those workloads. Therefore, as a final step of building our performance model, we evaluate the proposed hybrid ML-based solution through 6 rounds of generalizability testing. We utilize 6 test datasets comprising combinations of $Memory$, $Processors$, and *Payload Instance* that have not been encountered by the performance model during training or validation. The selection of the 6 test sets strategically introduces randomization into our testing scenarios, while also ensuring the inclusion of a diverse range of combinations for each variable. Additionally, each test set encompasses data across all rates, allowing us to extensively evaluate the model's generalizability in various parts of the input feature space. We qualify our performance prediction model by reporting the evaluation metrics defined in Section IV-C for the classifier and regressor outputs.

IV. Experiment Setup

A. Experiment Testbed

We have collected all our data by hosting TensorFlow Serving containers on a local server. The server is running with Ubuntu 22.04. It is equipped with 12 Intel(R) Xeon(R) CPU E5-2609 v3 processors running at 1.9 GHz and 66 GB of memory. We use TensorFlow Serving version 2.8.2 with Docker version 20.10.18 on this server to collect all our data. We use another machine that is connected to our server through an Ethernet link to set up our load generator. This machine also works with Ubuntu 22.04. It is equipped with 24 Intel(R) Xeon(R) CPU E5645 processors running at 2.4 GHz and 66 GB of available memory. We use *httperf* [20] version 0.9.1 as our load generator to send requests at customized rates to the server machine and collect the average response time for each experiment.

B. Data Collection

We emulate a realistic ML deployment scenario by varying workloads and resources simultaneously and switching services, i.e., DNNs, to collect response time data for different DNNs. We run 3 DNNs, namely DistilBERT, BERT, and a Convolutional Neural Network (CNN) model for classifying images from the MNIST dataset [21]. The last model is referred to as MNIST-CNN in this paper. The former two models are NLP models that classify text sentimentally, while MNIST-CNN classifies images. Each model is very different from the others in architecture, computation time, resource utilization, and input format.

Each data collection session runs for at least 1 minute, during which a DNN is served in a TensorFlow Serving container which is allocated a specific amount of processors and memory, and receives requests containing a specified quantity of payload instances at a specified rate from the load generator. The requests are sent at a specific rate with an exponential distribution of inter-arrival times. While we use the exponential distribution in our study, a distribution that reflects historical trends could be used by practitioners to train the predictive model in production scenarios. To collect our dataset, the processors are varied at three levels: low, medium, and high, which correspond to a quarter of the overall capacity (3 processors), half of the overall capacity (6 processors), and full capacity (12 processors), respectively. Similarly, memory is also varied at the same three levels i.e., low (11 GB), medium (22 GB), and high (44 GB). We vary the payload by adding 1, 2, or 3 data instances in the requests sent to the server, which we also refer to by the three levels of low, medium, and high, respectively. The default scenario in the experiments consists of requests with one payload instance served using the high memory and processor allocation. The rate varies from 1-100 requests per second for the NLP DNNs, but for the MNIST-CNN, it varies between $1-500$ requests per second because of its relatively low computational requirements. These rate ranges help gather experimental data at a variety of workloads and stress levels.

For classifying whether the DNN response time data will meet SLA stipulated response time targets under various workloads and resource settings, we define 5 targets of 0.5, 1, 10, 20, and 30 seconds. While 30 seconds might be a very high target, these response time targets were chosen to have a fair distribution of data samples that meet or exceed each target, since our dataset consists of many samples with very high response times observed under heavy workloads. For each target, we transform our dataset to include a new label column that contains 0 or 1 to indicate if or not the response time exceeds that target. On average, for the 5 response time targets across all 6 unseen test sets, we observe the response time targets being met in nearly 69% of the instances.

C. Evaluation Metrics and Primary Algorithms for Modeling

During the modeling process, we explore 8 ML algorithms for predicting DNN response time. These include Support Vector Machines, Random Forests, Decision Trees, Elastic, and Ridge Regressors, Adaboost, Gradient Boosting, and Extreme Gradient Boosting. All of the above ML algorithms were used to develop, tune, and evaluate response time prediction models using Python 3.10.12 in Google Colaboratory. We use an 80-20 split between training and test data for each model and explore experimentation of its hyperparameters to complete the preliminary evaluation of the suitability of each algorithm for our data. After the initial evaluation of these models on the collected data, we notice that Random Forest and Decision Tree Regressors are suitable algorithms for the data. Therefore, in Section V, we utilize Random Forest and Decision Tree Regression algorithms, fine-tuning them to achieve optimal prediction outcomes. Furthermore, for the classification task, we also solely concentrate on employing the Decision Tree and Random Forest classifiers.

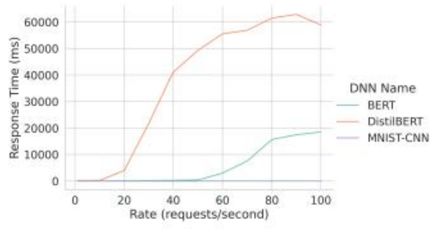
For evaluating our classification task, we report two primary metrics, *accuracy* and *f-score*, which are standard for evaluating ML classification and ensure correctness and unbiasedness. We aim to ensure high accuracy and f-score. For evaluating the regression task, we use the *Mean Absolute Percentage Error (MAPE)* metric. MAPE calculates the average relative error between the measured response time and the predicted response time across all 6 test cases.

V. Results

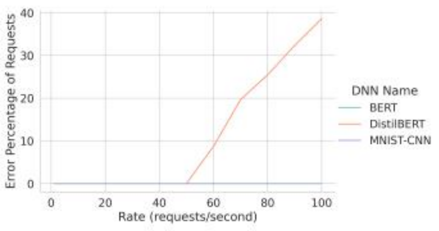
A. Impact of Experimental Features

We begin by closely studying the impact of each experimental feature on the 3 DNNs. Our goal is to discover the individual contribution of each feature and gain a comprehensive understanding of the relationships between these features.

1) Impact of Rate: The primary objective of our anticipated performance model is to predict the performance of a realistic deployment setup for DNNs that is subject to varying workloads throughout the day. We investigate the impact of the request rate as a key feature. Intuitively, a higher rate of incoming requests leads to longer response times due to increased queuing for server resources. From Figure 2, the response time for each DNN increases non-linearly with the request rate. Higher workloads can even cause the response time to decrease, as observed for DistilBERT in Figure 2(*a*). This occurs because the system becomes overloaded, causing many requests to be dropped. This acts as an admission control mechanism thereby decreasing the average response time, as demonstrated in Figure 2(*a* and *b*). All DNNs generally exhibit



(a) Response time vs. workload rate



(b) Request serving errors vs. workload rate

Fig. 2. Response times and error rates with varying workload

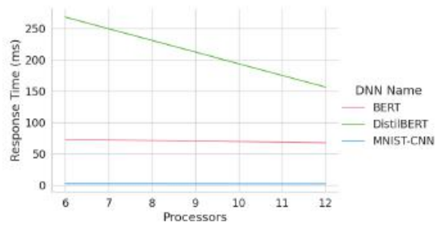
a similar pattern in the response time curve with increasing request rates, which suggests the potential for developing a unified performance model for multiple DNNs. However, there are significant variations in the response time of each

DNN under similar workloads and resource settings due to differences in resource consumption for individual requests. For example, at a rate of 1 request per second, MNIST-CNN has an average response time of 3.3 ms, while DistilBERT has an average response time of 106.7 ms under default feature values. Figure 2 demonstrates that by the time DistilBERT's response time starts to decrease due to system overload, MNIST-CNN's response time has not yet begun to escalate due to its significantly lower computational requirements. Furthermore, the default request size for MNIST-CNN is 5.4 kB, whereas the default request size for DistilBERT is 733 bytes. This indicates that the computational complexity of a model plays a more substantial role in determining its response time than the size of the input processed.

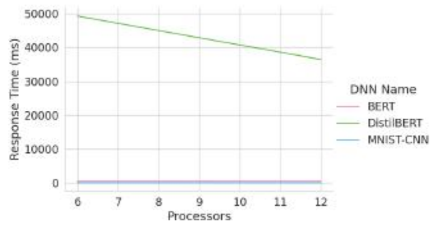
2) Impact of Processors: We now shift focus to the influence of the processors allocated to the TensorFlow Serving container on the response times of the hosted DNNs. Figure 3 presents the response times for different processor allocations, at rates of 10, 50, and 90 requests per second, respectively, with fixed memory and payload. As expected, increasing the number of processors assigned to the container leads to a reduction in response time. However, similar to our findings regarding the request rate, a linear increase in the processors does not result in a proportional decrease in response time. Moreover, the factor by which the response time is affected by the number of processors is not consistent across different rates. For example, at a rate of 10 requests per second, increasing the compute resources from 6 to 12 processors causes DistilBERT to decrease its response time by approximately 40% (from 250 to 150 ms). However, at a higher rate of 50 requests per second, the decrease in response time between 6 and 12 processors is only 20%, and at an even higher workload rate of 90 requests per second, response time only decreases by 15%.

Moreover, we observe that at a rate of 10 requests per second, when processors are doubled all three DNNs exhibit distinct slopes for response time decline, i.e., they decrease their response time by different factors. A similar observation holds for rates of 50 and 90 requests per second, where the higher workload amplifies the differences in behavior among the DNNs, thus demonstrating the complexity of the DNNs' responses to simultaneous changes in resources and workloads.

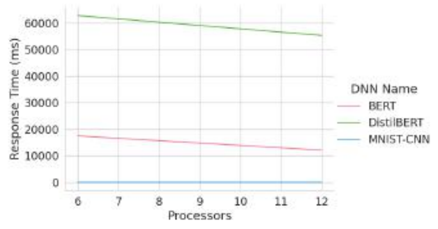
3) Impact of Memory: Memory allocation has a much smaller effect on response time compared to processors. This can be due to the fact that our experiments allocated a substantial amount of memory to the TensorFlow Serving container, with a maximum of 44 GB available. Even with a significant reduction in allocated memory, a considerable amount of memory is still available to the container. However, increasing the amount of memory does contribute to reducing the average response time, with the reduction becoming more pronounced under higher workloads. Additionally, DNNs exhibit distinct responses to changes in memory, and the disparities in response times among DNNs become more pronounced at higher workloads.



(a) 10 requests/second



(b) 50 requests/second



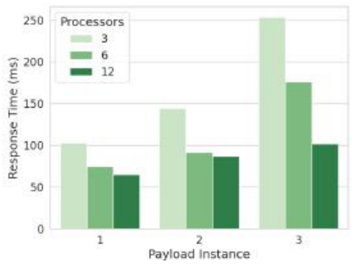
(c) 90 requests/second

Fig. 3. Response times of DNNs across processors at a fixed rate and memory

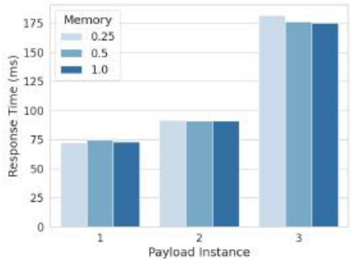
4) Impact of Payload Instance: We now examine the final variable in this study, which is the number of payload instances. Intuitively, as the number of payload instances in a request increases, the processing time per request will also increase, leading to longer response times. This is confirmed by Figure 4, which depicts the response time of BERT for 1, 2, and 3 payload instances at a rate of 10 requests per second. In Figure 4(a), the memory allocated to the serving container is fixed at medium allocation (22 GB), while in Figure 4(b), the number of processors is fixed at medium allocation of available processors (6). Once again, we observe that varying the number of processors has a more pronounced effect on the response time compared to varying memory. Additionally, we note that the change in response time is not directly proportional to the change in payload instances. Furthermore, the disparity in response times across different processor or memory values amplifies as the number of payload instances increases.

In summary, our analysis reveals the following key insights:

- While all DNNs demonstrate an increase in response time with growing workloads and decreasing resources, each DNN follows a unique trajectory, which is not directly proportional to workload increase or resource decrease.
- High workloads and resource constraints can lead to dropped requests, resulting in a deceptive decrease in the



(a) Memory fixed to half capacity



(b) Processors fixed to half capacity

Fig. 4. Response times of BERT across payload instances at fixed rate

TABLE I
RESULTS OF GENERALIZED REGRESSOR AND CLASSIFIER
(AC=ACCURACY, FS=F-SCORE)

Test Sets	**Average Accuracy Per Response Time Target (%)**					**Avg AC (%)**	**Avg FS (%)**
	0.5 s	**1 s**	**10 s**	**20 s**	**30 s**		
T1	97.72	97.72	100	97.72	100	98.63	98.23
T2	100	100	97.72	97.72	100	99.09	98.42
T3	96.23	96.23	96.23	96.23	100	96.98	96.43
T4	95.45	97.72	95.45	95.45	95.45	95.90	95.52
T5	100	100	97.72	95.45	97.72	98.18	97.92
T6	100	100	100	100	97.72	99.54	99.01

TABLE II
RESPONSE TIME PREDICTION ERRORS

Test Sets	**MAPE Per Response Time Target (%)**					**Average MAPE (%)**
	Target 0.5 s	**Target 1 s**	**Target 10 s**	**Target 20 s**	**Target 30 s**	
T1	6.48	6.49	6.49	6.49	6.23	6.44
T2	4.84	3.99	5.76	6.37	6.03	5.40
T3	9.78	20.30	17.46	22.02	26.86	19.28
T4	8.96	8.65	8.65	8.65	10.31	9.04
T5	6.97	6.65	7.95	8.03	7.74	7.47
T6	6.80	6.60	6.87	7.29	7.28	6.96
Avg.	**7.31**	**8.78**	**8.86**	**9.81**	**10.74**	**9.10**

response time curve at higher loads.

- All observations highlight the presence of a queuing mechanism caused by resource limitations and workload stress, leading to non-linear response time curves.

In light of these findings, it is clear that simple models are inadequate for capturing the behavior of different DNNs under varying resource-workload conditions.

B. Response Time Prediction

To perform the modeling tasks outlined in Section III, the classifier and regressor models are trained using identical datasets and evaluated using test samples that comprise 6 new previously unseen combinations of $Memory$, *Payload Instance*, and $Processors$. These test sets are carefully chosen to cover a wide spectrum of levels (low, medium, and high) for each of the three variables, and test combinations of these variables not encountered by either the classifier or the regressor during the training or validation phases.

Based on the aforementioned testing approach, the classifier exhibits an average accuracy of 98.05% across all 5 response time targets defined in Section IV-B. This high accuracy implies that one can confidently allocate appropriate resources using this classifier to ensure meeting response time targets based on the anticipated workload. Additionally, the classifier achieves an f-score of 97.59%, indicating that it is not benefiting from potential skews in training data. The results obtained on the 6 distinct test sets exhibit minimal variance, as depicted in Table I, demonstrating the consistent accuracy of the model across various resource-workload configurations.

After obtaining a resource allocation prescription from the classifier for an incoming workload, we can obtain a corresponding response time prediction through the regressor concatenated to the classifier with a MAPE of 9.10% across all considered response time targets during the experimentation. Table II shows the MAPE values observed on the 6 test sets against the 5 response time targets. Upon further analysis of the results obtained across the test sets, we are unable to track any particular pattern for certain resource configurations or workloads that result in higher prediction errors for all DNNs. However, we do note that generally, higher prediction errors result when the response time of a DNN is unusually high, and consequently, considerably unpredictable.

We also observe a correlation between the response time targets and the prediction errors observed against them. Notably, the MAPE exhibits a gradual increase as the response time target rises. This can be attributed to the fact that setting higher response time targets entails a greater likelihood of data samples with prolonged response times to be predicted by the classifier to meet the response time target, thereby increasing the high response time data samples used for response time prediction. Given the inherent difficulty in modeling higher response times for each DNN, the prediction error escalates for higher response time targets. These error trends can guide us in setting an upper limit for the response time target for each DNN task. For instance, if we aim for the prediction technique to maintain MAPE within 10% given the observations noted in Figure 5, it is imperative to set the response time target for each DNN below 30 seconds (or even below 20 seconds to allow a margin of error) and limit the incoming workloads such that SLA violations could be minimized.

Analyzing the individual classifier and regressor models used in the hybrid technique, we note that they adhere to similar patterns of feature importance. *DNN Name* and *Rate* are the most critical factors that help decide the response time. The classifier assigns the greater significance to *Rate* compared to *DNN Name*. The remaining three features play a relatively minor role in the decision-making process, with $Processors$ being the most influential, followed by *Payload Instance* and then $Memory$. Similarly, the regressor model follows a close pattern, albeit with the *DNN Name* being the most crucial feature, followed by *Rate*. This difference can be accounted for by the difference in the objectives of the classifier and the regressor. The response time of an individual request is primarily characterized by the DNN itself, while the workload rate influences the overall response time curve, as described in Section V-A. Therefore, by incorporating $Rate$ as the principal feature, closely accompanied by *DNN Name*, the classifier can facilitate the approximate binary determination of meeting the response time target effectively. However, the regressor aims to precisely model the response time in each scenario and thus chooses *DNN Name* as its primary feature for decision-making.

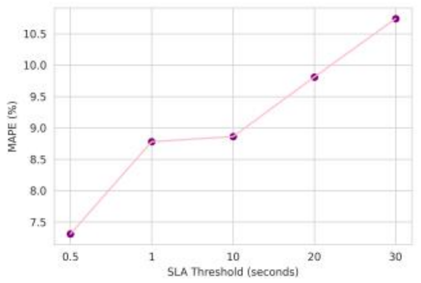


Fig. 5. MAPE observed for all included response time targets

VI. Conclusion

In this paper, we propose a unified hybrid ML-based performance model capable of providing predictions of response time target compliance and response time values for various types of DNNs operating under dynamically changing resource and workload conditions. Our proposed performance model utilizes features that are known prior to a DNN inference, eliminating the need for collecting metrics during request inference. We demonstrate the generalizability of our performance model to unseen workloads and resource configurations.

In future work, we aim to extend our model to offer predictions for new DNN models and hardware platforms, thus enabling predictions for inference requests related to unseen DNNs served on new hardware setups. These predictions are crucial when deploying new DNNs or DNN versions or when introducing new platforms for DNN deployment where extensive performance data is initially unavailable. We also plan to incorporate the influence of co-hosted DNNs within the same serving container on the observed response times for those DNNs.

References

[1] “The Top 10 Tech Trends In 2023 Everyone Must Be Ready For.” https://www.forbes.com/sites/bernardmarr/2022/11/21/the-top-10-tech-trends-in-2023-everyone-must-be-ready-for/.

[2] “Serving Models.” https://www.tensorflow.org/tfx/guide/serving/.

[3] “Use containers to Build, Share and Run your applications .” https://www.docker.com/resources/what-container/.

[4] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, “Application oriented dynamic resource allocation for data centers using docker containers,” *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2017.

[5] “Runtime options with Memory, CPUs, and GPUs.” https://docs.docker.com/config/containers/resource$_constraints/$.

[6] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, “Integer quantization for deep learning inference: Principles and empirical evaluation,” *arXiv preprint arXiv:2004.09602*, 2020.

[7] S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy, “On the use of ml for blackbox system performance prediction,” in *NSDI*, 2021.

[8] B. Cai, Q. Guo, and X. Dong, “Autoinfer: A self-driving management for resource efficient, slo-aware machine learning inference in gpu clusters,” *IEEE Internet of Things Journal*, 2022.

[9] Y. Li, Z. Han, Q. Zhang, Z. Li, and H. Tan, “Automating cloud deployment for deep learning inference of real-time online services,” in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1668–1677, IEEE, 2020.

[10] Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, and J. Gonzalez, “Train big, then compress: Rethinking model size for efficient training and inference of transformers,” in *International Conference on machine learning*, pp. 5958–5968, PMLR, 2020.

[11] D. Khudia, J. Huang, P. Basu, S. Deng, H. Liu, J. Park, and M. Smelyanskiy, “Fbgemm: Enabling high-performance low-precision deep learning inference,” *arXiv preprint arXiv:2101.05615*, 2021.

[12] Y. Li, D. Zeng, L. Gu, Q. Chen, S. Guo, A. Zomaya, and M. Guo, “Lasagna: Accelerating secure deep learning inference in sgx-enabled edge cloud,” in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 533–545, 2021.

[13] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, “Multi-model machine learning inference serving with gpu spatial partitioning,” *arXiv preprint arXiv:2109.01611*, 2021.

[14] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, “Mlperf inference benchmark,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459, IEEE, 2020.

[15] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, and C.-J. Wu, “The vision behind mlperf: Understanding ai inference performance,” *IEEE Micro*, vol. 41, no. 3, pp. 10–18, 2021.

[16] P. Ross and A. Luckow, “Edgeinsight: Characterizing and modeling the performance of machine learning inference on the edge and cloud,” in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1897–1906, IEEE, 2019.

[17] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, “Latency and throughput characterization of convolutional neural networks for mobile computer vision,” in *Proceedings of the 9th ACM Multimedia Systems Conference*, pp. 204–215, 2018.

[18] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, “Serving dnns like clockwork: Performance predictability from the bottom up,” *arXiv preprint arXiv:2006.02464*, 2020.

[19] C. Zhang, M. Yu, W. Wang, and F. Yan, “Enabling cost-effective, slo-aware machine learning inference serving on public cloud,” *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1765–1779, 2020.

[20] D. Mosberger and T. Jin, “httperf—a tool for measuring web server performance,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[21] “MNIST Dataset.” https://www.kaggle.com/datasets/hojjatk/mnist-dataset .