Towards A Flexible Accuracy-Oriented Deep Learning Module Inference Latency Prediction Framework for Adaptive Optimization Algorithms

Jingran Shen[1], Nikos Tziritas[2], and Georgios Theodoropoulos[3]

[1] Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, P.R. China
petershen815@126.com

[2] Department of Informatics and Telecommunications, University of Thessaly, Lamia, Greece
nitzirit@uth.gr

[3] Department of Computer Science and Engineering, and Research Institute for Trustworthy Autonomous Systems, Southern University of Science and Technology (SUSTech), Shenzhen, P.R. China
theogeorgios@gmail.com

Abstract. With the rapid development of Deep Learning, more and more applications on the cloud and edge tend to utilize large DNN (Deep Neural Network) models for improved task execution efficiency as well as decision-making quality. Due to memory constraints, models are commonly optimized using compression, pruning, and partitioning algorithms to become deployable onto resource-constrained devices. As the conditions in the computational platform change dynamically, the deployed optimization algorithms should accordingly adapt their solutions. To perform frequent evaluations of these solutions in a timely fashion, RMs (Regression Models) are commonly trained to predict the relevant solution quality metrics, such as the resulted DNN module inference latency, which is the focus of this paper. Existing prediction frameworks specify different RM training workflows, but none of them allow flexible configurations of the input parameters (e.g., batch size, device utilization rate) and of the selected RMs for different modules. In this paper, a deep learning module inference latency prediction framework is proposed, which i) hosts a set of customizable input parameters to train multiple different RMs per DNN module (e.g., convolutional layer) with self-generated datasets, and ii) automatically selects a set of trained RMs leading to the highest possible overall prediction accuracy, while keeping the prediction time / space consumption as low as possible. Furthermore, a new RM, namely MEDN (Multi-task Encoder-Decoder Network), is proposed as an alternative solution. Comprehensive experiment results show that MEDN is fast and lightweight, and capable of achieving the highest overall prediction accuracy and R-squared value. The Time/Space-efficient Auto-selection algorithm also manages to improve the overall accuracy by 2.5% and R-squared by 0.39%, compared to the MEDN single-selection scheme.

Keywords: Deep learning · Machine learning · Latency Prediction.

1   Introduction

1.1   Background

There has been an increasing number of cloud/edge-based applications that utilize DNN (Deep Neural Network) models to improve task execution efficiency as well as decision-making quality. Specifically, large DNN models like GPT-3 [3] and SAM [5] (Segment Anything Model) are facing extra deployment concerns due to their complex structures and heavy model sizes. The memory deficiency problem becomes even more critical when these models need to be deployed to a resource-constrained edge environment. To this end, a large amount of research on different categories of model optimization algorithms has been conducted through the past few years. These algorithms include i) compressing DNN models through quantization techniques or autoencoders [1,15], ii) pruning DNN models by reducing model parameters or by modifying model structures [7,8,15,18], and iii) partitioning DNN models onto multiple devices [1,8,9,11,15,17,18]. Table 1 lists several adaptive model optimization algorithms as examples.

Table 1. Adaptive DNN Model optimization algorithms.

Algorithm	Compression	Pruning	Partitioning	Description
Auto-Split [1]	✓		✓	two-way partitioning with module-wise quantization
PTEENet [7]		✓		adaptive early-exiting
MoDNN [11]			✓	Linear module weight parititioning
k-way DAG Partitioning [9]			✓	k-way partitioning
Edgent [8]		✓	✓	two-way partitioning and adaptive early-exiting
Branchy GNN [15]	✓	✓	✓	two-way partitioning, adaptive early-exiting, and intermediate feature compression

Occasionally, the varying available memory and network bandwidth of the devices may affect the quality of the currently provided solution. Hence, model optimization algorithms should adjust their solutions accordingly to adapt to these environment changes, which means that solution quality evaluations will happen frequently. To perform the evaluations in a timely fashion due to real-time requirements, RMs (Regression Models) are commonly trained to predict relevant solution metrics - specifically in this paper, the resulted module inference latency. These trained RMs can also be applied to other adaptive optimization algorithms like inference request batching [6,16] and input data partitioning [4,11,19]. Several prediction frameworks [8,10,20] have been introduced to

specify different RM training workflows, but none of them allow flexible configurations of the input parameters (e.g., batch size, FLOPS, device utilization rate) and the of selected RMs for different DNN modules. Since different modules have varying degrees of structure and computation complexity, they require different regression models and input parameters for prediction. Hence, a flexible prediction framework that can accommodate such diversity is essential.

1.2 Gap Analysis on Related Work

In this paper, a DNN module inference latency prediction framework is proposed to address the aforementioned problem by i) hosting a set of customizable input parameters to train multiple different RMs per DNN module (e.g., convolutional layer) with self-generated datasets, and ii) automatically selecting a set of trained RMs leading to the highest possible overall prediction accuracy, while keeping the prediction time / space consumption as low as possible. Besides, a new RM named MEDN (Multi-task Encoder-Decoder Network) is proposed as an alternative solution.
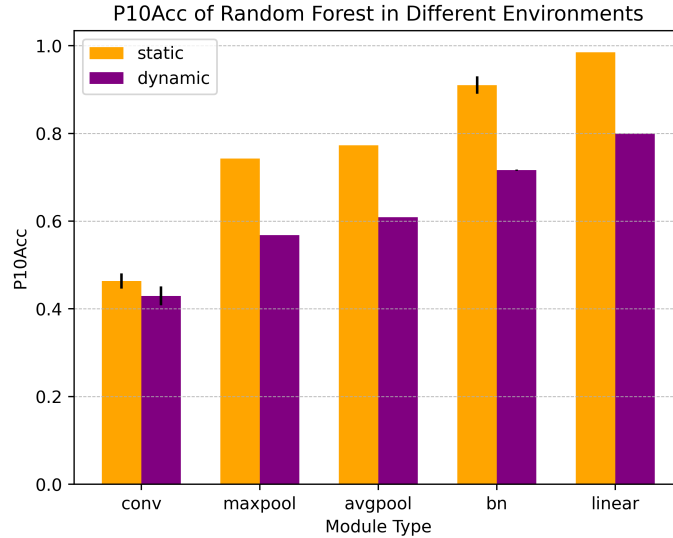

Fig. 1. P10-Accuracy of Random Forest in different environments.

Training a universal RM for all modules [10, 12] is technically more difficult and non-scalable due to the ever-increasing number of deep-learning modules. Instead, the proposed framework focuses on training one RM per module and forming a group of prediction experts through RM auto-selection, while existing frameworks only support the single-selection scheme that uses one RM for all

modules. Furthermore, the proposed framework measures the available memory and utilization rate of the device to further improve the prediction accuracy in a dynamic environment, where other workloads are also consuming device resources. Other frameworks like Edgent [8] and nn-Meter [20] do not cover this scenario. As shown in Figure 1, random forests trained and refined by nn-Meter face significant accuracy drops when switching to a dynamic environment. Additionally, nn-Meter does not handle cases where the data batch size is more than 1. Finally, the proposed framework introduces Inferable Parameters to calculate more meaningful input features (e.g., data/module size) as extra information for RMs to handle, which is not applied by Edgent [8]. In addition, the proposed framework introduces a newly designed encoder-decoder network - MEDN (Multi-task Encoder-Decoder Network), which is faster on predictions and far more light-weight than random forests utilized by nn-Meter. Besides, MEDN better captures features from the input parameters and generally outperforms other Deep RMs (e.g., Multi-Layer Perceptron, namely MLP [10,12]) on both accuracy and R-squared metrics. Table 2 summarizes the features of the proposed framework compared to other relevant latency prediction frameworks.

Table 2. Gap analysis of relevant researches on the proposed topic.

Features	Edgent [8]	nn-Meter [20]	SWFU [10]	Proposed Framework
Utilize RM	✓	✓	✓	✓
Handle larger batch sizes	✓		✓	✓
Infer extra information		✓	✓	✓
Measure device dynamics			✓	✓
Utilize Deep RM			✓	✓
Use Encoder-Decoder				✓
Support RM Auto-selection				✓

The rest of the paper is structured as follows: Section 1 discusses the motivation for introducing the proposed framework and briefly states its advantages compared to other existing DNN module inference latency prediction frameworks. Section 2 outlines the key components and operations of the proposed framework. Section 3 presents the experimental frame and a quantitative analysis and evaluation of the proposed system. Section 4 summarizes the paper and lists potential future research directions.

2   Framework Design

The key objective of the proposed framework is to support flexible RM training configurations as well as accuracy-oriented automatic RM selections. The follow-

ing components described in this section constitute the ability of the proposed framework to achieve the stated objective.

2.1   Parameters

The input parameters are categorized into three genres, namely i) Sampling Parameters, ii) Measurable Parameters, and iii) Inferable Parameters, to help with systematic analysis on the their importance in a dynamic environment, where other workloads on the device affect the module inference performances.

Table 3. Parameters for each module.

Module	Sampling Parameters	Measurable Parameters	Inferable Parameters
avgpool	N, C_i, L, K, S	M, U	N_d
bn	N, L, C_i	M, U	N_d, N_m
conv	N, L, C_i, C_o, K, S	M, U	N_d, N_m
linear	N, C_i, C_o	M, U	N_d, N_m
maxpool	N, C_i, L, K, S	M, U	N_d

The Sampling Parameters refer to the original parameters that can be used to construct the corresponding data and module. As an example, a convolutional module can be constructed by specifying the Sampling Parameters including the input/output channels $C_i, C_o \in [3, 512]$, kernel size $K \in \{1, 3, 5, 7, 9\}$, and stride $S \in \{1, 2, 4\}$. The corresponding input data for the convolutional module can be constructed by specifying the batch size $N \in [1, 64]$, input height/weight $L \in \{224, 112, 56, 32, 28, 27, 14, 13, 8, 7\}$ along with the input channels C_i. Apart from these original parameters, the available memory M (in bytes) and utilization rate $U \in [0, 1.0]$ of the device should be recorded as Measurable Parameters to capture the features from the dynamic environment. Intuitively, RMs have to be informed about whether the device is free and available to commit enough resources to module inference tasks. When the device is heavily occupied with other running jobs, RMs should predict a relatively slower inference speed than the normal scenario (i.e., static environment). Finally, Inferable Parameters describe the set of parameters whose values can be calculated from other Sampling/Measurable/Inferable Parameters. This category of parameters generally offers fine-grained information to help RMs better understand the input features. As an example, the number of model weight parameters N_m indicates the module complexity and the input data size N_d reflects the computation overheads.

Depending on the structure of different RMs, different parameter sets can be configured flexibly during the training procedure. Table 3 summarizes the available parameters for various modules including Convolution, Max/Average Pooling, Batch Normalization (abbreviated as "bn" in this paper), and Linear.

2.2   Regression Model Training Workflow

Figure 2 demonstrates the RM training workflow of the proposed framework. Basically, training data samples are constructed by packing all configured parameter values as input features. In detail, Sampling Parameter values are randomly generated in the specified ranges and then used to construct the corresponding data as well as modules for ground-truth latency measurement. Next, device status is evaluated into Measurable Parameters and all Inferable Parameters are calculated from the existing parameters. Finally, the packed parameter values and the measured latency results are combined into a dataset, which is then used to train a set of pre-configured RMs. As the last step of the training workflow, an auto-selection algorithm outputs a set of RMs achieving the highest possible overall prediction accuracy, while keeping the prediction time / space consumption as low as possible.
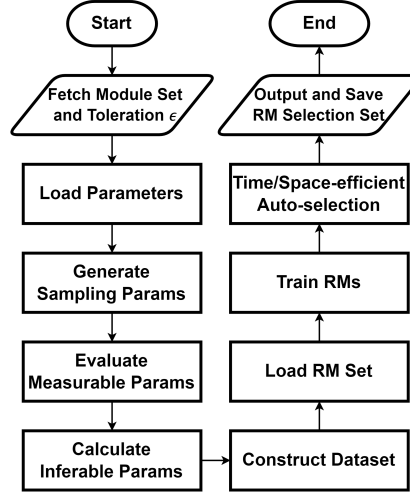

Fig. 2. RM training workflow of the framework.

2.3   Towards a New Regression Model

A new regression model, namely MEDN (Multi-task Encoder-Decoder Network) is designed as an alternative solution for RM selection. Essentially, the model is composed of one encoder and two decoders, as shown in Figure 3. The encoder compresses the input features x into the latent space and the prediction decoder utilizes the latent feature to produce the predicted inference latency as the output $\hat{y}$. Additionally, a reconstruction decoder is applied to rebuild the input features as its output $\hat{x}$, using the latent feature from the encoder. The

reconstruction task aligns with the idea of autoencoders [2], which aim to formulate a more meaningful understanding of the input features. The training losses of the reconstruction task and prediction task are weighted sum into one single loss for the backpropagation procedure.

For the experimental analysis in this paper, MLPs are applied as the underlying structure for both the encoder and the decoders. The structure of the reconstruction decoder is symmetric to that of the encoder, and the hidden dimensions in the prediction decoder gradually drop along the power of twos. Nevertheless, it is possible to specify a different MEDN configuration as long as the basic idea preserves the same. Theoretically, MEDN is more capable of capturing the input features and providing more accurate predictions, compared to simpler deep RMs like MLP. MEDN is also faster and lighter than Random Forest, whose inference speed and model size are easily affected by the size and complexity of the dataset.
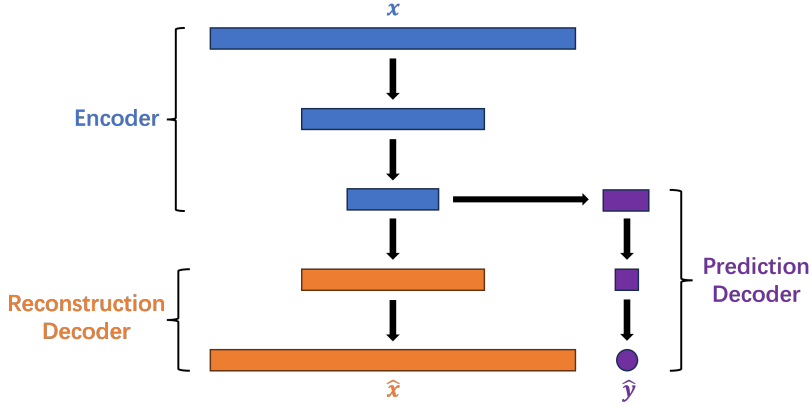

Fig. 3. MEDN structure.

2.4   Auto-Selection

Algorithm 1 illustrates how the framework manages to find an RM selection set automatically regarding the prediction accuracy and R-squared metric results. In addition, the algorithm examines an extra objective (time-per-sample/model-size) to further improve the performance in the time/space dimension. Specifically, the algorithm first filters a set of RMs possessing the highest prediction accuracy results (Line 5-6). RMs with less than ϵ_a accuracy difference (as a tolerance factor) from the one with the highest accuracy are also included. Then, a similar filtering operation is performed on the R-squared metric (Line 7–8). Finally, one RM is selected according to the specified extra objective (Line 9),

where the selected RM demonstrates the least inference time or model size. Utilizing filtering with tolerance, the Time/Space-efficient Auto-selection algorithm ensures that the overall accuracy and R-squared results are at the highest possible level, while attempting to place preferences on faster or lighter RMs. It is pertinent to note that when two RMs possess the same prediction accuracy, the one with a higher R-squared value is preferred. Intuitively, a higher R-squared value attempts to confirm that falsely predicted samples have smaller absolute errors, thus leading to a better result. Hence, the filtering operation is performed on accuracy first and R-squared afterwards.

Algorithm 1 Time/Space-efficient Auto-selection.

Require: Evaluation result set Y; accuracy and R-squared tolerance ϵ_a, ϵ_r
Ensure: Selection S: module $m \Rightarrow$ regression model u
 1: $o \leftarrow$ time-per-sample (time-efficient) or model-size (space-efficient)
 2: $S \leftarrow \emptyset$
 3: **for** each module $m \in M$ **do**
 4:     $Y_m \leftarrow \{y \in Y \mid y.\mathrm{m} = m\}$
 5:     $y_1 \leftarrow \arg \max_{y \in Y_m} y.\mathrm{acc}$
 6:     $Y_1 \leftarrow \{y \in Y_m \mid y_1.\mathrm{acc} - y.\mathrm{acc} \leq \epsilon_a\}$
 7:     $y_2 \leftarrow \arg \max_{y \in Y_1} y.\mathrm{r}$
 8:     $Y_2 \leftarrow \{y \in Y_1 \mid y_2.\mathrm{r} - y.\mathrm{r} \leq \epsilon_r\}$
 9:     $y_s \leftarrow \arg \min_{y \in Y_2} y.\mathrm{o}$
10:     $S[m] \leftarrow y_s.\mathrm{u}$
11: **end for**
12: **return** S

---

3 Experimental Evaluation

3.1 Configuration

In total, four RMs including the proposed MEDN, Random Forest (abbreviated as RF), Multi-layer Perceptron (abbreviated as MLP), as well as Linear Regression (abbreviated as LR) are trained and evaluated on nine modules, four of which are composite modules. The configurations of Random Forests and Multi-layer Perceptrons are respectively the same as nn-Meter [20] and SWFU [10]. In total, 10,000 samples are generated for convolutional modules and 2,000 samples are generated for other modules. The datasets are split into train, validation, test set with the ratio of $7 : 1 : 2$. The input features are min-max scaled before being utilized by the RMs. All presented results are evaluated on test sets. To mimic a dynamic environment, random module inference jobs are occasionally generated on the device.

Table 4 lists the configurations of MEDN, including the hidden dimensions of the encoder (excluding the input dimension) and the weight ratio which is

Table 4. Configurations of MEDN.

Module	Encoder Hidden	Weight Ratio (Reconstr : Pred)
avgpool	[64, 32, 16]	0.001
bn	[32, 16, 8]	1.0
conv	[128, 64, 32]	100.0
linear	[64, 32, 16]	0.01
maxpool	[64, 32, 16]	0.01

exactly reconstruction weight divided by prediction weight. All deep RMs are trained for 500 epochs with the learning rate as 0.005. All losses are calculated using the Smooth L1 Loss function. The repeated experiments are implemented with PyTorch [13] as well as scikit-learn [14] and conducted on Tesla P100 PCIe.

3.2   Results on Single-RM Selection

According to Table 5, other RMs like RF and MLP achieve lower overall prediction accuracy and R-squared compared to the proposed MEDN. Besides, they are generally less efficient both in time (time-per-sample as "Tps" in milliseconds) and space dimension (model-size as "Size" in kilobytes). The overall metric results are essentially percentage differences with MEDN, averaged over all modules. The results prove that MEDN serves better as a alternative RM solution on all evaluated metrics.

With LR, only the Linear module is characterized by a high prediction accuracy, indicating that there is potentially a simple linear relationship between the input features and the output latency for this module. However, the R-squared results of RF and MLP turn negative on the Linear module. According to the detailed experiment result, the standard deviation of the dataset for this module is rather small. Therefore, R-squared results generally become negative when the Root Mean Squared Error (RMSE) values of RF (0.0355) and MLP (0.021) exceed the dataset standard deviation value (0.0207), while the proposed MEDN achieves a positive R-squared with an RMSE value of 0.0203.

On the other hand, all RMs struggle with the latency prediction of convolutional modules due to the complicated module structures they possess. Nevertheless, MEDN manages to outperform other RMs by increasing the reconstruction task weight, as shown in Table 4. Besides, composite convolutional modules are generally easier to predict than the monolithic convolutional module, due to potential kernel fusion optimization techniques that are able to simplify the computation dynamics. Surprisingly, the prediction results on pooling layers are generally far from ideal. One of the reasons might be that these modules do not own weight parameters, thus lacking enough inferable information for deep RMs to comprehend. With a more complex model structure, especially like MEDN, this can easily lead to overfitting. Future research should consider adding more

Table 5. Results on different regression models for multiple deep-learning modules.

Module\RM	MEDN				RF			
	Acc	R^2	Tps	Size	Acc	R^2	Tps (ms)	Size (KB)
avgpool	0.6700	0.9792	0.4148	40	0.7625	0.9457	83.4442	46089
bn	0.8775	0.9937	0.8246	21	0.9050	0.9904	50.4511	23595
bn+relu	0.8300	0.9750	0.7895	21	0.8300	0.9934	51.6820	23620
conv+bn	0.6915	0.7726	0.4649	110	0.5925	0.7034	114.5584	342353
conv+bn+relu	0.6920	0.7495	0.6129	110	0.5795	0.6895	147.0191	342261
conv	0.6695	0.8626	0.4591	110	0.5875	0.7570	116.1598	342490
conv+relu	0.6925	0.9618	0.9540	110	0.6165	0.8489	137.5373	342270
linear	0.9600	0.0328	0.4048	40	0.8850	-1.6179	76.6457	41028
maxpool	0.6425	0.8745	0.4207	40	0.7425	0.7124	83.5899	46100
Avg diff% **vs. MEDN**	-	-	-	-	**-2.5%**	**-24%**	**+1.8e4%**	**+2.0e5%**
Module\RM	MLP				LR			
	Acc	R^2	Tps	Size	Acc	R^2	Tps (ms)	Size (KB)
avgpool	0.5600	0.9035	0.5509	77	0.0950	0.1603	0.0732	3
bn	0.8800	0.9945	0.6687	77	0.2400	0.7110	0.1714	3
bn+relu	0.8275	0.9907	0.4295	77	0.2325	0.3918	0.1564	3
conv+bn	0.5900	0.7399	1.0352	77	0.1355	0.0460	0.0589	3
conv+bn+relu	0.6370	0.6304	0.3112	77	0.1545	0.0342	0.0924	3
conv	0.4855	0.8823	0.3959	77	0.1280	0.0617	0.1001	3
conv+relu	0.5755	0.9110	0.3680	77	0.1540	0.0523	0.1083	3
linear	0.9550	-0.0297	0.3075	77	0.9575	0.0578	0.0727	3
maxpool	0.5775	0.8937	0.7146	77	0.0875	0.0393	0.1477	3
Avg diff% **vs. MEDN**	**-7.1%**	**-3.2%**	**+1.4%**	**+77%**	**-50%**	**-63%**	**-81%**	**-93%**

Inferable Parameters like the expected FLOPS (floating-point operations per second).

3.3   Results on Ablation Experiment for MEDN

To examine the effect of the reconstruction decoder in MEDN, an ablation experiment is performed to evaluate a non-reconstruction version (MEDN-Direct as MEDN-D) for comparison. Besides, MEDNs trained on different parameter sets are evaluated. MEDN-No-Infer (MEDN-XI) and MEDN-No-Measure (MEDN-XM) handle all parameters except the Inferable/Measurable Parameters. MEDN-RAW (MEDN-R) handles only Sampling Parameters.

Figure 4 shows the evaluated metric results averaged over all modules. P20-Accuracy is the default accuracy metric in other result tables, specifying the ratio of samples whose prediction percentage error is within 20%. First, it can be noticed that MEDN outperforms MEDN-D on all metrics, verifying that the reconstruction decoder serves an important role in the improved performance.
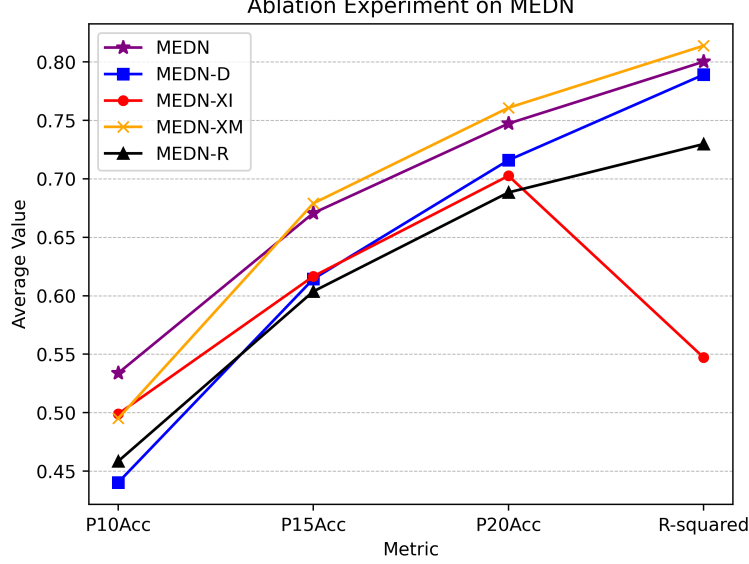
Fig. 4. Ablation experiment on MEDN.

Next, MEDN-XI performs generally worse than MEDN-XM but better than MEDN-R, indicating that Inferable Parameters provide more helpful guidance to latency prediction than Measurable Parameters, although both lead to better accuracy results. Finally, it is discovered that Measurable Parameters allow MEDN to be more accurate on the correctly predicted samples, resulting in higher P10-Accuracy than that of MEDN-XM.

3.4   Results on Auto-Selection

Table 6 records the resulted RM selection sets of the Time-efficient Auto-selection algorithm (time-per-sample as the extra objective) and Space-efficient Auto-selection algorithm (model-size as the extra objective). Both algorithms set $\epsilon_a = \epsilon_r = 0.05$, specifying the tolerance as 5%. Compared to the MEDN single-selection scheme where MEDN is selected for all modules, both selection sets manage to achieve a higher overall prediction accuracy and R-squared value. In detail, MEDN variants with the highest P20-Accuracy results are selected for the convolutional modules, and RF is selected for the max pooling layer where MEDN does not perform ideally. LR is selected for the simple Linear module, which results in a significant decrease in the prediction time and storage space.

"Results on diff" in the table record the original metric results of the RMs that differ among the schemes. Since both algorithms prioritize accuracy, the selected RMs for the modules are rather similar except for one difference on

the composite Batch Normalization + ReLU module. Examining the difference, it can be discovered that for the Time-efficient Auto-selection algorithm, the inference time per sample of the MLP on the module is lower than that of MEDN-XM by the Space-efficient Auto-selection algorithm, while the model size is larger. This aligns with their extra objectives to save time and space respectively.

Table 6. Results on auto-selection schemes compared to MEDN single-selection.

Module/Scheme		Time-efficient Auto-selection	Space-efficient Auto-selection
avgpool		MEDN-XM	MEDN-XM
bn		MEDN-XM	MEDN-XM
bn+relu		MLP	MEDN-XM
conv+bn		MEDN-R	MEDN-R
conv+bn+relu		MEDN-XM	MEDN-XM
conv		MEDN-XM	MEDN-XM
conv+relu		MEDN-XM	MEDN-XM
linear		LR	LR
maxpool		RF	RF
Results on diff	Tps (ms)	**0.4295**	**0.4820**
	Size (KB)	**77**	**20**
Avg diff% vs. MEDN	Acc	**+2.4%**	**+2.63%**
	R^2	**+0.41%**	**+0.36%**

4 Conclusion

In this paper, a flexible accuracy-oriented DNN module inference latency prediction framework is proposed to support customizable parameter configurations and accuracy-oriented RM auto-selection in a dynamic environment. A newly designed RM, namely MEDN, is also proposed as an alternative RM solution. The quantitative analysis presented in this paper demonstrates that MEDN is fast and lightweight, and capable of achieving the highest overall prediction accuracy and R-squared value. The Time/Space-efficient Auto-selection algorithm also manages to improve the overall accuracy by 2.5% and R-squared by 0.39%, compared to the MEDN single-selection scheme. In the future, more Inferable Parameters and Measurable Parameters will be utilised in the experimental analysis in order to fully exploit the feature comprehension ability of MEDN.

SUSTech Research Institute for Trustworthy Autonomous Systems, China. Corresponding author: Georgios Theodoropoulos.

References

1. Banitalebi-Dehkordi, A., Vedula, N., Pei, J., Xia, F., Wang, L., Zhang, Y.: Autosplit: A general framework of collaborative edge-cloud ai. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. p. 2543–2553. KDD '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3447548.3467078, https://doi.org/10.1145/3447548.3467078
2. Bank, D., Koenigstein, N., Giryes, R.: Autoencoders (2021)
3. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners (2020)
4. Hu, C., Li, B.: Distributed inference with deep learning models across heterogeneous edge devices. In: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications. pp. 330–339 (2022). https://doi.org/10.1109/INFOCOM48880.2022.9796896
5. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., Dollár, P., Girshick, R.: Segment anything. arXiv:2304.02643 (2023)
6. Kum, S., Oh, S., Yeom, J., Moon, J.: Optimization of edge resources for deep learning application with batch and model management. Sensors **22**(17) (2022). https://doi.org/10.3390/s22176717, https://www.mdpi.com/1424-8220/22/17/6717
7. Lahiany, A., Aperstein, Y.: Pteenet: Post-trained early-exit neural networks augmentation for inference cost optimization. IEEE Access **10**, 69680–69687 (2022). https://doi.org/10.1109/ACCESS.2022.3187002
8. Li, E., Zeng, L., Zhou, Z., Chen, X.: Edge ai: On-demand accelerating deep neural network inference via edge computing (2019)
9. Lin, P., Shi, Z., Xiao, Z., Chen, C., Li, K.: Latency-driven model placement for efficient edge intelligence service. IEEE Transactions on Services Computing **15**(2), 591–601 (2022). https://doi.org/10.1109/TSC.2021.3109094
10. Liu, G., Dai, F., Huang, B., Li, L., Wang, S., Qiang, Z.: Towards accurate latency prediction of dnn layers inference on diverse computing platforms. In: 2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). pp. 1–7 (2022). https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927862
11. Mao, J., Chen, X., Nixon, K.W., Krieger, C., Chen, Y.: Modnn: Local distributed mobile computing system for deep neural network. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. pp. 1396–1401 (2017). https://doi.org/10.23919/DATE.2017.7927211
12. Mendoza, D.: Predicting latency of neural network inference (2020)

13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Curran Associates Inc., Red Hook, NY, USA (2019)
14. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Curran Associates Inc., Red Hook, NY, USA (2019)
15. Shao, J., Zhang, H., Mao, Y., Zhang, J.: Branchy-gnn: a device-edge co-inference framework for efficient point cloud processing (2023)
16. Shi, W., Zhou, S., Niu, Z., Jiang, M., Geng, L.: Multiuser co-inference with batch processing capable edge server. IEEE Transactions on Wireless Communications **22**(1), 286–300 (2023). https://doi.org/10.1109/TWC.2022.3192613
17. Tang, X., Chen, X., Zeng, L., Yu, S., Chen, L.: Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence. IEEE Internet of Things Journal **8**(12), 9511–9522 (2021). https://doi.org/10.1109/JIOT.2020.3010258
18. Teerapittayanon, S., McDanel, B., Kung, H.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 328–339 (2017). https://doi.org/10.1109/ICDCS.2017.226
19. Zeng, L., Chen, X., Zhou, Z., Yang, L., Zhang, J.: Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. IEEE/ACM Transactions on Networking **29**(2), 595–608 (2021). https://doi.org/10.1109/TNET.2020.3042320
20. Zhang, L.L., Han, S., Wei, J., Zheng, N., Cao, T., Yang, Y., Liu, Y.: Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. p. 81–93. MobiSys '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3458864.3467882, https://doi.org/10.1145/3458864.3467882