Predicting Inference Time and Energy Consumption in Deep Learning Using MLPs

Chengjin Lyu, Mohsen Nourazar and Bart Goossens
TELIN-IPI, Ghent University – imec
St-Pietersnieuwstraat 41, B-9000 Gent, Belgium

***Abstract*—With the rapid evolution of deep learning technologies, the efficient deployment of models in real-time and energy-sensitive environments has become increasingly vital. Accurately predicting how deep models consume resources ensures that these models operate within the constraints of their intended scenarios. This is particularly essential for applications such as Neural Architecture Search (NAS), edge computing, and distributed systems. Most current work focuses on predicting layer-wise inference time and energy consumption, but the summed predictions often do not align with the actual runtime and energy cost. This paper introduces a framework using two-stage Multi-Layer Perceptrons (MLPs) to predict model-wise inference time and energy consumption of deep neural networks. The first stage of our approach involves layer-wise inference time and energy usage predictions tailored to various types of neural network layers. Building upon these initial outputs, the second-stage predictor employs another MLP to aggregate these layer-wise estimations into a comprehensive prediction of the overall model's performance. We validate the effectiveness of the proposed method on two real computing platforms. This framework can enhance the design and deployment of deep learning architectures, by accurately estimating model-wide inference time and energy consumption.**

***Index Terms*—deep learning, model inference, time prediction, energy consumption**

I. Introduction

In recent years, deep learning has achieved notably high performances in various tasks, such as image recognition [1], medical diagnostics [2], and autonomous driving [3]. Usually, these deep learning technologies rely on various Deep Neural Networks (DNNs) with high computational demands. Real-world scenarios with constrained computational resources such as edge computing [4]–[6] and distributed neural network inference [7]–[9] typically require that these DNNs be deployed with a trade-off between high inference speed and low energy consumption. To design more sustainable and efficient DNNs, recent Neural Architecture Search (NAS) methods [10]–[12] explicitly consider inference constraints (e.g., inference time and energy consumption) as hard guides during the architecture design phase.

In practice, measuring the inference time and energy consumption of DNNs can be time-consuming and resource-draining, especially when dealing with different hardware setups and operational conditions [13]. These measurements require extensive testing across diverse devices and software configurations, significantly delaying large-scale deployments. Moreover, designing a DNN architecture on one kind of end device using NAS methods could generate a vast amount of candidate DNNs in the search stage, making it inefficient to measure them physically. Therefore, it is essential to have predictive approaches to help engineers simulate the DNNs' inference speed and energy efficiency, ensuring that their DNNs run within the optimal bounds of restricted computational resources [14].

There have been many studies [15]–[17] on predicting layer-wise inference time, while it is known that the simple sum of layer-wise predictions is not equivalent to the actual model-wise inference performance. Furthermore, there is limited work that focuses on the predictions of both inference time and energy consumption. Addressing these two aspects together highlights a significant opportunity for efficient design and sustainable deployment of DNNs in resource-sensitive environments. Inspired by these observations, we propose a framework to predict both time and energy consumption during the inference phase for DNNs, leveraging two-stage Multi-Layer Perceptrons (MLPs). The predictive model can be trained not only on GPUs but also on a variety of emerging DNN hardware accelerators, such as FPGAs, DPUs, and TPUs, which are becoming increasingly diverse. The main contributions of this work are summarized as follows:

- A predictive framework based on two-stage MLPs is proposed to predict both inference time and energy consumption for DNNs. It can provide valuable insights for optimizing design and efficient deployment.
- The model-wise MLP predictor aggregates the layer-wise MLP predictions to avoid the inaccuracy problem associated with the direct sum of these predictions.
- The size of the trained MLP models is small, making them effective for dynamic load balancing and edge deployment applications.
- Experimental evaluation is conducted for several typical neural networks on datasets collected from two computing platforms. The results confirm the effectiveness of our proposed framework.

The remainder of this paper is organized as follows. Section II covers the background and motivation. Section III describes the proposed method in detail. Section IV presents the experiments and results. Section V concludes the paper and proposes

This research was realized within the context of the FARAD2SORT ICON project, funded by Flanders Make, the strategic research Centre for the Manufacturing Industry in Belgium. In addition, this work received funding from the Flemish Government (Flanders AI Research Program).

potential future work.

II. Background and Motivation

In this section, we review the related research by dividing it into two categories: layer-wise prediction and model-wise prediction. Additionally, we outline the motivation for our proposed work by identifying the gaps in current research and discussing how our approach addresses these shortcomings.

A. Layer-Wise Prediction

Layer-wise prediction methods focus on predicting individual layers' inference time and/or energy consumption within a DNN. As for the entire DNN model's estimation, these methods utilize the sum of all predictions to represent it.

In the tasks of model acceleration and NAS, analytical methods have been widely used to achieve the optimal speed-accuracy trade-off. To estimate the layer-wise inference time, the number of Floating Point Operations (FLOPS) is applied in [15]. Similarly, the research in predicting the energy consumption of DNNs in the inference stage also explores the usage of FLOPS [18]. Lahmer et al. [19] propose an empirical model based on the number of Multiply-Accumulate Operations (MAC) for predicting energy consumption within certain layers. Moolchandani et al. [20] build an analytical model for performance in distributed training of transformers. A comprehensive study of the analytical models for performance prediction on GPUs is given in [21]. This work focuses on kernel-level prediction, where a kernel is a small unit of code executed. In practice, each layer in the DNN is mapped to one or more CUDA kernels, depending on the complexity and the type of operations required. Thus, the kernel divisions across devices might vary. In our work, we treat each layer as a basic computation unit, making it adaptive to various heterogeneous computing platforms.

However, the number of computational operations used in these analytical methods does not correspond simply to the actual energy consumption or inference time [22]. The inference performance is influenced not only by computation cost but also by many other factors. Specifically, the related factors include the GPU clock speed, GPU resource utilization, communication overheads, algorithmic optimizations, and the efficiency of the DNN framework.

To achieve accurate predictions, many researchers focus on modeling the layer parameters using learning-based approaches. NeuralPower [23] employs a learning-based polynomial regression method to perform layer-wise predictions. However, each layer type requires a careful selection of polynomial terms, leading to limited flexibility and scalability. In [22], linear regression models per layer type are constructed to predict both inference time and energy consumption. A recent work [17] adopts neural networks to perform layer-wise inference time predictions for four types of DNN layers and sum them up to get the model-wise inference time estimation for better collaborative computing.

B. Model-Wise Prediction

The category of model-wise research accounts for factors such as network topology, layer arrangement, and layer interdependencies. Typically, complex techniques based on machine learning or deep learning are employed to train model-wise predictors. There is less research on predicting inference performance at the model level compared to the extensive work done on layer-wise prediction.

BRP-NAS [12] presents a model-wise predictor based on Graph Convolutional Networks (GCNs), outperforming the layer-wise prediction methods. This work confirms that an accurate inference time predictor can improve the quality of NAS significantly. Nevertheless, the graph-based method heavily relies on the structures of DNNs in its training set.

Zhang et al. [24] presents a systematic inference time prediction solution named nn-Meter by dividing the DNN inference into kernels that represent the fusion of several layers. These kernels are detected automatically from a dataset with 14 layer types, and the random forests are used as kernel-wise predictors. The summation of all kernel-wise predictions forms the final predicted model inference time. This method works on a kernel-by-kernel basis rather than a layer-by-layer basis. As a result, the training procedure is more elaborate as a large diversity of computation kernels and their parameters need to be covered.

C. Motivation

In general, the layer-wise prediction methods are based on the assumption that either inference time or energy consumption of each layer in a DNN is independent of other layers. However, a DNN's complexity and runtime optimization make the execution time on real hardware not strongly correlated to the simple hierarchical addition. Thus, a summing operation does not lead to an accurate inference performance prediction. As mentioned above, the model-wise predictive approaches show some promising results when considering the entire model. However, these methods do not provide insights into the performance of individual layers, and the generalization ability is not guaranteed when the training data does not cover a wide enough range of scenarios.

We propose a predictive framework based on two-stage MLPs to address these challenges. The first stage is layer-wise prediction using MLPs to model layer parameters, without requiring expert expertise to build complex mathematical models. The second-stage MLPs refine and aggregate these predictions from layer levels by capturing nonlinear relationships among various layers. The final output is an accurate model-wise estimation, and the predictions for individual layers are also available. In this way, this method can be used to guide not only the search of neural networks in NAS but also the splitting of DNNs in distributed inference. As this method is based on neural networks, it can be easily adapted to new GPU architectures and edge devices while maintaining high accuracy.

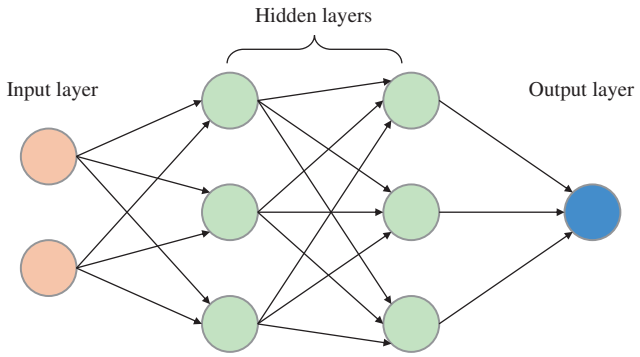


Fig. 1. A simple MLP with two hidden layers.

III. Proposed Method

This section introduces a two-stage predictive framework based on MLPs to predict inference time and energy consumption, as well as the creation of a benchmark dataset. The following pivotal insight guides our work: although various DNNs deployed on diverse devices normally could have a wide range of inference time and energy consumption, it is still feasible to predict these two key values of inference on the targeted device using the DNN parameters [22]. To this end, we first estimate the layer-wise inference time and energy consumption using MLPs. Then, another MLP serves as a model-wise predictor to accumulate the layer-wise predictions. This two-stage predictive framework fully utilizes MLP's capability of learning nonlinearities within sampled data to perform accurate prediction.

A. Multi-Layer Perception

The multi-layer perception has been one of the simplest and most common neural networks for a long time in the world. It maps inputs to outputs through a feed-forward manner and gets trained via the back-propagation mechanism to reduce output errors. An MLP has one input layer, one output layer, and at least one hidden layer between them. The nodes in an MLP are called neurons, and each neuron conducts a weighted sum of its input connections. Typically, every neuron in the hidden layers is followed by a non-linear activation function, allowing MLP to capture the non-linear relationships between inputs and outputs.

A simple MLP with two hidden layers is shown in Fig. 1. This MLP has two neurons in the input layer, representing that the input data has two features. The output layer contains only one neuron, which means there is only one output for each input data. Each hidden layer consists of three neurons in this simple MLP, while there is strict no restriction on the configuration of hidden layers. In practice, MLPs with multiple hidden layers belong to a classic type of DNNs, capable of handling complex computational tasks. Specifically, all the MLPs used in this work are configured with three hidden layers.

B. Dataset Collection

To cover the representative layer types that might be used in a DNN model, we create a large dataset with 36 different kinds of layers in this work. The collected dataset includes not only Conv, FC and pooling layers that are popular in smaller datasets but also various widely used activation and normalization layers. To the best of our knowledge, this is the first dataset consisting of such a wide range of DNN layer types to perform inference time and energy consumption prediction.

To train an accurate predictor, it is vital to have correct measurements of inference time and energy cost as ground truths. However, it is relatively difficult to get an accurate layer-wise measurement. For inference time, PyTorch provides tools for profiling the layer-wise runtime for both CUDA and CPU operations, while neither of their accumulated values over all layers of a DNN model matches the actual model-wise inference time. This may be attributed to the complex sources impacting the inference time, including CUDA/CPU operations, memory accesses, communication costs, and profiling overheads. Even though, these two measurements reflect the core CUDA and CPU operational time and can be beneficial to understand the running of a DNN on a target device. To customize a profiler, a common practice is to insert a hook to each layer to measure the layer-wise inference time (i.e., to capture the execution time of multiple computational kernels within one layer), but it introduces additional profiling overhead to the overall measurement. Thus, both PyTorch and custom layer-wise measurements do not correspond well to the real model inference time.

In this work, we introduce a novel profiling method that measures layer-wise inference time with high accuracy. This profiler executes the DNN model incrementally from the first layer to the last, recording cumulative execution time at each step. The profiler derives the inference time for each layer by slicing these cumulative measurements. This ensures that the sum of all layer times matches the actual runtime of the entire model, addressing the limitations of existing profiling techniques. Although this design minimizes overhead compared to PyTorch profilers and the custom hook-based method, it may involve some noise among layers due to memory caching effects and processing unit stalling. Thus, we include the three measurements mentioned above in our dataset as auxiliary attributes and use them in our proposed predictor to predict the model-wise inference time.

To measure the energy consumption, we assume that the power consumption of a GPU executing the inference of a single neural network layer is approximately constant during the inference following the work [19]. This assumption is due to the uniform workload distribution across GPU cores and fixed computational requirements of a certain layer. We profile the power usage in PyTorch using the NVIDIA Management Library (NVML) to monitor the layer-wise power in watts. For the nth layer of a DNN, we compute the layer-wise energy

consumption E_n as:

$$E_n = P_n \cdot T_n, \tag{1}$$

where T_n is the layer inference time and P_n is the power value. As illustrated in [19], the energy consumption of sequential neural network layers equals the sum of their energy consumption. Thus, the total energy cost of the whole DNN model is defined as the sum of all layer-wise energy consumption:

$$E_{total} = \sum_{n=1}^{N} E_n. \tag{2}$$

Similar to inference time measurement, we compute three auxiliary energy consumption values. In this way, we collect both model runtime and energy consumption during the test phase for each DNN model in our dataset.

C. Two-Stage Predictor

In this work, both time and energy predictors share the same design of MLPs. The first part of our framework is to predict inference time and energy consumption for each type of layer in DNNs. To predict over a large scope of data, we select MLPs to extract the complex relationships between the parameters of a layer and its measured runtime and energy cost.

For each layer in a DNN, a 16-dimensional feature is constructed based on its layer parameters, as shown in Fig. 2a. This feature vector includes layer type information, input shape, output shape, and layer configurations. The layer type is set as an integer ranging from 0 to 29, and the other parameters also vary accordingly. For convolutional layers, the features from input and output shapes contain channels, widths, heights, and batch size, while the key layer configurations are defined by kernel size, stride, padding, and dilation. For each of the other layer types, we extract the features by parsing the layer parameters to fit into the 16-dimensional vectors.

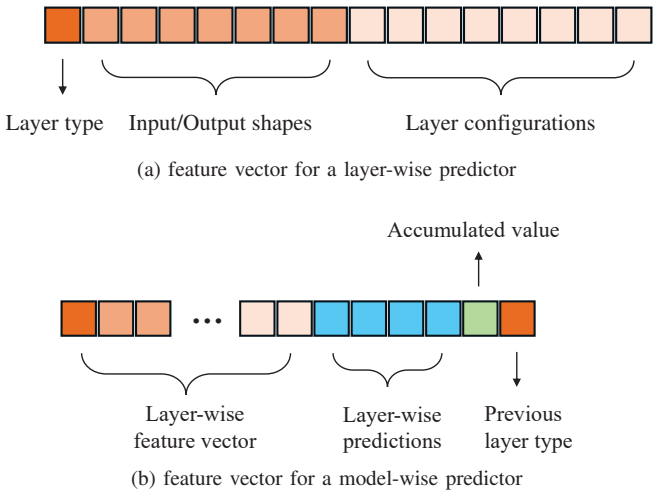
To enhance the predictor's learning ability, we configure an MLP with four outputs in a multi-task learning manner, where one main output is assisted with three auxiliary outputs. For each layer's predictions, three auxiliary outputs are related to CUDA operations, CPU operations, and layer running with overhead. For each k-th output, where $k \in \{1, 2, 3, 4\}$, the loss L_k is computed using Mean Absolute Error (MAE) as follows:

$$L_k = \frac{1}{N_s} \sum_{i=1}^{N_s} |y_{k,i} - \hat{y}_{k,i}|, \tag{3}$$

where N_s represents the number of samples in a training batch, $y_{k,i}$ and $\hat{y}_{k,i}$ denotes the ground truth value and predicted value, respectively. The overall loss function L_{total} combines the four individual losses L_k, weighted by their respective coefficients λ_k:

$$L_{\text{total}} = \sum_{k=1}^{4} \lambda_k L_k. \tag{4}$$

In the experiments, we set all the weights λ_k to 1. This configuration leverages the auxiliary outputs to encourage the predictor to learn more generalized features and capture the complex patterns within the data. By focusing on shared representations across tasks, the model is encouraged to generalize beyond the training data, enhancing its ability to perform well on unseen data.



(a) feature vector for a layer-wise predictor

(b) feature vector for a model-wise predictor

Fig. 2. Example of feature vectors used in the proposed two-stage prediction process. (a) In the first stage, predictions are made using layer-wise feature vectors that include layer types, input/output shapes, and layer configurations. (b) In the second stage, the model-wise predictor incorporates not only these layer-wise feature vectors but also the outputs from the first stage, accumulated main output values, and the types of previous layers.

Furthermore, we proceed to perform the model-wise prediction using another MLP by aggregating all the outputs from the layer level. As shown in Fig. 2b, the feature vector constructed for layer-wise prediction is concatenated with the layer-wise predictions, an accumulated value of the main outputs, and an integer indicating the previous layer type, resulting in a 22-dimensional input feature vector for the model-wise predictor. The accumulated value is calculated as the summation of the main predicted values over all previous layers. The ground truth corresponding to this input feature vector is the cumulative measurement from the first layer to the current layer.

With this unique design, the second-stage MLP predictor is capable of refining and aggregating the layer-wise predictions instead of simply summing them. In the test stage, the predictor processes the input data layer by layer in a series, and the predicted value of the last layer in a DNN is seen as the model-wise prediction. Additionally, it is very handy to get the inference time and energy consumption prediction for a certain combination of layers because the predicted cumulative values are available. Thus, our framework can be integrated to guide the DNN deployment in distributed systems.

In this work, the MLPs used for layer-wise prediction have 16 input neurons, 4 output neurons, and 3 hidden layers. The model-wise predictor is an MLP with 22 input neurons, 1 output neuron, and 3 hidden layers. The implementation details of MLPs can be found in Section IV.

TABLE I
DATA COLLECTION PLATFORMS

Platform	GPU	CPU	Framework
2080Ti	RTX 2080 Ti	Intel Core i7-9800X	PyTorch 2.2
3080Ti	RTX 3080 Ti	AMD Ryzen 9 7950X	PyTorch 2.2

IV. EXPERIMENTS

In this section, we describe the dataset used in our experiments, the evaluation metric, and the implementation details. The experimental results for both inference time and energy consumption predictions are presented and discussed.

A. Dataset

Two platforms with different setups of CPUs and GPUs are used to get the measurements in our dataset, as listed in Table I. Based on these two platforms, 90 variants of typical DNN models are loaded for inference with random inputs with sizes ranging from 224×224 to 780×780. Moreover, 300 synthetic models are generated by randomly connecting the feasible layers to enhance the dataset. Specifically, the inference time and energy consumption refer to the corresponding values of a DNN model evaluated on the target platform with one input. When profiling the DNNs, we collect a measurement as the average of 100 inference runs.

In a real-world scenario, a robust predictor is supposed to be able to perform predictions for unseen DNN models. Here we refer to an unseen model as one whose family is not included in the training set, but all its layer types are available to support the training. Thus, the network structure of an unseen model is not learned during the training stage. To this end, we leave out a set of DNNs as a test set and exclude those models from our predictors' training phase. Thus, the predictor is tested to validate if it can predict accurately for unseen models. Table II summarizes the DNN model families in our test set, including GoogLeNets [25], Inceptions [26], ShuffleNets [27], and MnasNets [10], MobileNets [28], and EfficientNets [29], where each model family may contain various variants. For instance, the family of EfficientNets is a collection of 11 different variants from two versions of EfficientNets. It is interesting to see that the inference time of MnasNets on 2080Ti platform remains almost constant (between 6.4 and 6.5 milliseconds) while the energy consumption on the same platform varies from 284.7 to 839.0 millijoules. This confirms that the energy cost prediction is a non-trivial task and not directly related to the inference time. Moreover, the MnasNets' maximal inference time on the 3080Ti platform is more than three times greater than the minimal value (4.7 versus 1.4 milliseconds), representing the heterogeneous designs among hardware platforms. It is worth noting that NAS techniques are employed to boost the DNN design in MnasNets, MobileNets, and EfficientNets, while the first three belong to the traditional hand-crafted DNNs. In short, the collected dataset covers a wide spectrum of inference time and energy consumption from diverse DNNs.

TABLE II
MEASURED INFERENCE TIME AND ENERGY CONSUMPTION ON TEST SET

Models	2080Ti		3080Ti	
	Time (ms)	Energy (mJ)	Time (ms)	Energy (mJ)
GoogLeNets	8.4 - 8.5	445.8 - 915.0	2.1 - 4.0	329.7 - 820.6
Inceptions	14.2 - 14.8	816.3 - 1948.4	3.4 - 8.2	588.4 - 2003.2
ShuffleNets	7.9 - 8.2	372.6 - 684.1	1.9 - 2.7	229.7 - 486.4
MnasNets	6.4 - 6.5	284.7 - 839.0	1.4 - 4.7	169.4 - 964.6
MobileNets	6.7 - 8.2	303.7 - 745.9	1.5 - 3.6	187.4 - 724.6
EfficientNets	10.4 - 44.6	538.2 - 4262.6	2.5 - 24.1	373.7 - 5899.7

B. Evaluation Metric

Mean Absolute Percentage Error (MAPE) is utilized as an evaluation metric. A lower MAPE value means that the predicted values are closer to the ground truths. MAPE expresses the predicting error as a percentage, making it intuitive to understand the average errors across different scales. For our experiments, this scale invariance of a metric is essential due to the large scope of the dataset.

C. Implementation Details

All the MLPs used in this work contain three hidden layers, with 256, 512, and 512 neurons, respectively. At the training stage, we randomly shuffle all the layer-wise training data to prevent the predictor from overfitting and enhance the generalization ability. ReLu is used as a non-linear activation function. A predictor is trained with a batch size of 512 for 400 epochs using AdamW optimizer to minimize the L1 loss function. We set the learning rate to 0.001 for layer-wise MLPs and 0.01 for model-wise MLPs. An MLP trained in this work contains around 0.4 million parameters and 0.8 million FLOPS, making it suitable for deployment on resource-constrained devices.

D. Experimental Results

Here, we present the experimental results for predicting inference time and energy consumption on two platforms for various DNNs using our two-stage predictive framework. Furthermore, we compare our results with the direct sum of layer-wise predictions to evaluate the effectiveness of the proposed method.

For convenience, we named our method as MLP-MLP and the direct sum as MLP-SUM, respectively. As illustrated in Table III, our MLP-MLP method outperforms MLP-SUM on both 2080Ti and 3080Ti platforms in terms of inference time prediction. We observe similar results for energy consumption prediction in Table IV, where MLP-MLP delivers better predictions compared to MLP-SUM.

This confirms that the usage of second-stage MLPs can aggregate layer-wise predictions into model-wise estimations smoothly. It is also interesting to see that the prediction errors for the NAS-based DNNs (i.e., MnasNets, MobileNets, and EfficientNets) are smaller overall than the hand-crafted ones. This is possibly due to less unnecessary complexities in these DNNs fully or partially designed by NAS. On the other hand,

TABLE III
MAPE OF INFERENCE TIME PREDICTIONS

Models	2080Ti		3080Ti	
	MLP-SUM	MLP-MLP	MLP-SUM	MLP-MLP
GoogLeNets	21.68%	19.05%	43.12%	28.15%
Inceptions	23.20%	25.53%	38.73%	30.28%
ShuffleNets	24.70%	6.90%	73.56%	15.10%
MnasNets	18.17%	5.02%	38.22%	14.62%
MobileNets	18.23%	2.16%	50.89%	8.02%
EfficientNets	17.61%	9.62%	46.22%	14.75%
All	19.37%	8.52%	51.67%	15.40%

TABLE IV
MAPE OF ENERGY CONSUMPTION PREDICTIONS

Models	2080Ti		3080Ti	
	MLP-SUM	MLP-MLP	MLP-SUM	MLP-MLP
GoogLeNets	18.17%	10.23%	39.33%	24.53%
Inceptions	30.45%	13.13%	35.45%	24.32%
ShuffleNets	14.56%	13.77%	62.56%	39.81%
MnasNets	4.80%	13.50%	25.52%	17.63%
MobileNets	18.51%	5.49%	43.07%	10.17%
EfficientNets	14.51%	10.12%	38.32%	15.21%
All	14.02%	11.32%	42.24%	22.72%

this means a good predictor can also be integrated into the NAS to precisely guide the design.

V. CONCLUSION

In this paper, we introduce a predictive framework utilizing MLPs to estimate both inference time and energy consumption for DNNs. This framework features a novel two-stage MLP design that aggregates and refines layer-wise predictions. We validate the effectiveness of the proposed method through experiments conducted on datasets collected on two computing platforms covering the representative DNN layer types. By accurately predicting inference time and energy consumption, researchers can optimize the balance between model performance and efficiency, enhancing both the design and deployment of deep neural networks for sustainable artificial intelligence. In the future, we will continue to improve our predictive framework and investigate its potential across more hardware platforms, particularly edge devices with limited computational resources.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 770–778.

[2] R. Aggarwal, V. Sounderajah, G. Martin, D. S. Ting, A. Karthikesalingam, D. King, H. Ashrafian, and A. Darzi, "Diagnostic accuracy of deep learning in medical imaging: A systematic review and meta-analysis," *NPJ Digital Medicine*, vol. 4, no. 1, p. 65, 2021.

[3] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen, and A. Y. Ding, "A survey on approximate edge ai for energy efficient autonomous driving services," *IEEE Communications Surveys & Tutorials*, vol. 25, pp. 2714–2754, 2023.

[4] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[5] J. Shao and J. Zhang, "Communication-computation trade-off in resource-constrained edge inference," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 20–26, 2020.

[6] M. Yuan, L. Zhang, F. He, X. Tong, M.-H. Song, Z. Xu, and X.-Y. Li, "Infi: End-to-end learning to filter input for resource-efficiency in mobile-centric inference," *IEEE Transactions on Mobile Computing*, vol. 23, pp. 3523–3538, 2024.

[7] A. Thomas, Y. Guo, Y. Kim, B. Aksanli, A. Kumar, and T. S. Rosing, "Hierarchical and distributed machine learning inference beyond the edge," in *Proceedings of the International Conference on Networking, Sensing and Control*. IEEE, 2019, pp. 18–23.

[8] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8099–8110, 2020.

[9] K. Liu, C. Liu, G. Yan, V. C. S. Lee, and J. Cao, "Accelerating dnn inference with reliability guarantee in vehicular edge computing," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 3238–3253, 2023.

[10] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2019, pp. 2820–2828.

[11] X. Dong, L. Liu, K. Musial, and B. Gabrys, "NATS-Bench: Benchmarking nas algorithms for architecture topology and size," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[12] Ł. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "Brp-nas: prediction-based nas using gcns," in *Proceedings of the International Conference on Neural Information Processing Systems*, 2020, pp. 10 480–10 490.

[13] 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 *Proceedings of the Annual International Symposium on Computer Architecture*. IEEE, 2020, pp. 446–459.

[14] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2017, pp. 5687–5695.

[15] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *Proceedings of the IEEE Conference on Computer Communications*, IEEE. IEEE, 2020, pp. 854–863.

[16] O. M. Alqahtani and L. M. Ramaswamy, "A layer decomposition approach to inference time prediction of deep learning architectures," in *Proceedings of the IEEE International Conference on Machine Learning and Applications*. IEEE, 2022, pp. 855–859.

[17] G. Liu, F. Dai, X. Xu, X. Fu, W. Dou, N. Kumar, and M. Bilal, "An adaptive dnn inference acceleration framework with end–edge–cloud collaborative computing," *Future Generation Computer Systems*, vol. 140, pp. 422–435, 2023.

[18] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "Delight: Adding energy dimension to deep neural networks," in *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 112–117.

[19] S. Lahmer, A. Khoshsirat, M. Rossi, and A. Zanella, "Energy consumption of neural networks on NVIDIA edge boards: an empirical model," in *Proceedings of the International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks*, 2022, pp. 365–371.

[20] D. Moolchandani, J. Kundu, F. Ruelens, P. Vrancx, T. Evenblij, and M. Perumkunnil, "Amped: An analytical model for performance in distributed training of transformers," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2023, pp. 306–315.

[21] J. Lemeire, J. G. Cornelis, and E. Konstantinidis, "Analysis of the analytical performance models for gpus and extracting the underlying pipeline model," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 32–47, 2023.

[22] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "Previous: A methodology for prediction of visual inference performance on iot devices," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9227–9240, 2020.

[23] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Proceedings of the Asian Conference on Machine Learning*. PMLR, 2017, pp. 622–637.

[24] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2021, pp. 81–93.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015, pp. 1–9.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 2818–2826.

[27] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 116–131.

[28] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2019, pp. 1314–1324.

[29] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.