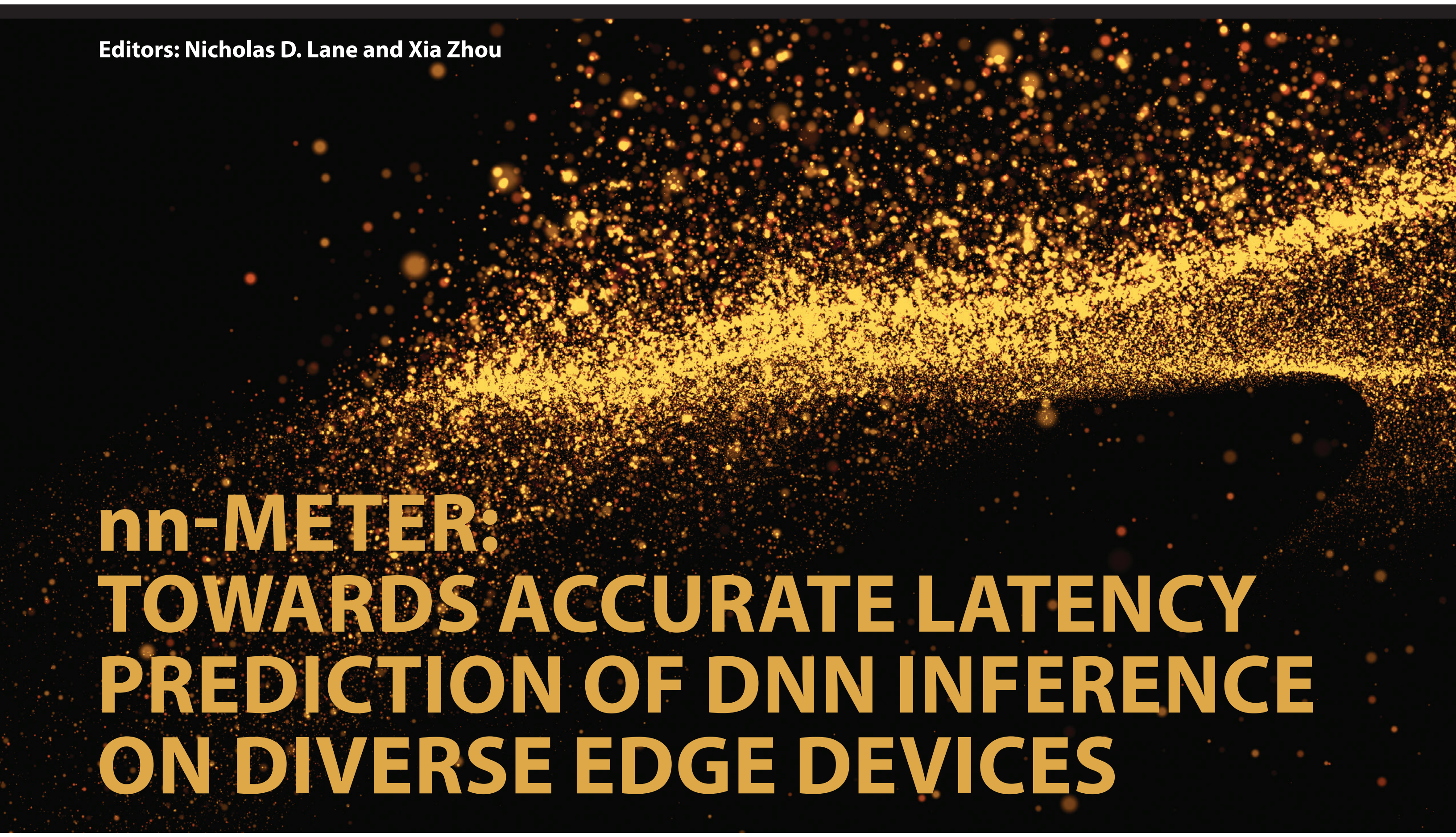
Li Lyna Zhang *Microsoft Research* **Shihao Han** *Microsoft Research, Rose-Hulman Institute of Technology*
Jianyu Wei *Microsoft Research, University of Science and Technology of China*
Ningxin Zheng, Ting Cao and Yuqing Yang *Microsoft Research*
Yunxin Liu *Institute for AI Industry Research (AIR), Tsinghua University*

Editors: Nicholas D. Lane and Xia Zhou



nn-METER: TOWARDS ACCURATE LATENCY PREDICTION OF DNN INFERENCE ON DIVERSE EDGE DEVICES

Excerpted from "nn-Meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices" from *MobiSys '21: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* with permission. https://dl.acm.org/doi/10.1145/3458864.3467882 ©ACM 2021

Inference latency has become a crucial metric in running Deep Neural Network (DNN) models on various mobile and edge devices. To this end, latency prediction of DNN inference is highly desirable for many tasks where measuring the latency on real devices is infeasible or too costly. Yet it is very challenging and existing approaches fail to achieve a high accuracy of prediction, due to the varying model-inference latency caused by the runtime optimizations on diverse edge devices. In this paper, we propose and develop *nn-Meter*, a novel and efficient system to accurately predict the DNN inference latency on diverse edge devices. The key idea of nn-Meter is dividing a whole model inference into *kernels*, i.e., the execution units on a device, and conducting kernel-level prediction. nn-Meter builds atop two key techniques: (i) *kernel detection* to automatically detect the execution unit of model inference via a set of well-designed test cases; and (ii) *adaptive sampling* to efficiently sample the most beneficial configurations from a large space to build accurate kernel-level latency predictors. nn-Meter achieves significant high prediction accuracy on four types of edge devices.

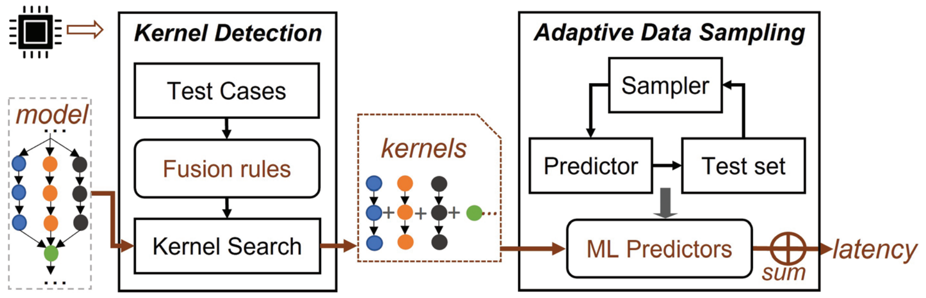


FIGURE 1. System architecture of nn-Meter. It offline detects fusion rules and builds machine learning predictors of kernels.

DNNs have been widely used in today's mobile and edge applications [1]. In many applications, such as on-device video analytics, face recognition, AR/VR, etc., DNN models are constrained by efficiency constraints (e.g., latency). To design a model with both high accuracy and efficiency, model compression [2,3] and the recent Neural Architecture Search (NAS) [4,5] consider the inference latency of DNN models as the hard design constraint.

However, measuring the inference latency for DNN models is laborious and expensive. It requires developers to perform a deployment process on the physical device to obtain the latency. The engineering effort is tremendous for diverse edge devices (e.g., mobile CPU/GPU and various AI accelerators) and different inference frameworks (e.g., TFLite and OpenVINO). Even on a single device, it may be extremely time-consuming to measure a large number of models in NAS tasks (e.g., ProxylessNAS [4] explores ~0.3 millions of models in just one round of search). Such a high cost can hinder the scalability and make the measurement-based method practically infeasible to support the fast-growing number of edge devices.

Consequently, approaches have been proposed to predict the inference latency. For example, the FLOPs (i.e., the number of multiply-adds) based method has been widely applied to evaluate the efficiency [3,6], which is simple but not a direct metric of latency. To predict a model latency, many NAS works [4,5] build the operator-wise lookup table. Such operator-level methods sum up the latencies of all operators. However, they do not consider the model latency differences caused by runtime optimizations of model graphs. For instance, many frameworks merge multiple operators into one fused operator to accelerate the inference, which impacts the inference latency significantly. Recently, the state-of-the-art BRP-NAS [7] uses graph convolutional networks (GCN) to predict latency of the NASBench201 [8] dataset on various devices. It captures the runtime optimizations by learning the representation of model graphs and corresponding latency. However, this model-graph based approach depends heavily on the tested model structures and may not work for many unseen model structures.

In this work, we propose and develop a novel system called nn-Meter that aims to accurately predict the latency of *arbitrary* DNN models on diverse edge devices. The key idea of nn-Meter is dividing a whole model inference into multiple *kernels* that are independent execution units of the model inference on a device. A kernel may be either a single primitive operator or a fusion of multiple operators, depending on the runtime and hardware. nn-Meter builds latency predictors for kernels and predicts the total latency of a model by the latency sum of all kernels of the model. We implement and evaluate nn-Meter on four popular platforms of edge devices: mobile CPU, mobile Adreno640 GPU, mobile Adreno630 GPU and Intel VPU (a representative AI accelerator for edge devices). Significantly, nn-Meter achieves a prediction accuracy of 99.0%, 99.1%, 99.0%, 83.4% on the CPU, Adreno640 GPU, Adreno630 GPU and VPU, respectively.

In this article, we first introduce the high-level system design of nn-Meter, then we present our two key components: kernel detection and adaptive data sampling. Finally, we report the evaluation results to demonstrate the effectiveness of nn-Meter.

nn-METER DESIGN

One key design choice of nn-Meter for high prediction accuracy is to conduct kernel-level prediction. This design choice is based on two observations. First, kernel is the basic scheduling and execution unit (e.g., GPU kernels) in deep-learning frameworks, particularly on edge devices. Thus, the notion of kernel naturally captures the diverse runtime optimizations including *operator fusion*, the most important optimization that can largely impact the latency. Second, despite a very large number of DNN models, the kinds of operators and kernels are stable with a relatively small set. Any models are just different combinations of operators/kernels. Therefore, kernel-level prediction is generic enough to support unseen new models.

Figure 1 illustrates the system architecture of nn-Meter. nn-Meter employs two core components to realize accurate latency prediction for a DNN model: Kernel Detection and Adaptive Data Sampling. Conceptually, the former automatically divides the target model into a set of kernels, and the latter samples the most beneficial configurations from a large space to build accurate kernel-level latency predictors. For each kernel of a given model, we extract the features and predict its latency. Then, nn-Meter sums up all the predicted kernel latencies as the whole-model latency.

KERNEL DETECTION

Challenges

The first challenge of nn-Meter is how to split a model into a proper set of kernels on various edge devices. Due to the diverse runtime optimizations, the executed kernels are varying on different devices. For example, the Conv+add is a fused kernel on a mobile GPU, but not on a mobile CPU or Intel VPU. Furthermore, many inference frameworks are not open-sourced. Even for the open-sourced ones, it requires runtime expertise to determine the kernels.

To address this challenge, nn-Meter employs a kernel detector that automatically detects the possible kernels on various edge devices in a black-box matter. We design a set of test cases to detect whether two operators can be fused or not. A DFS

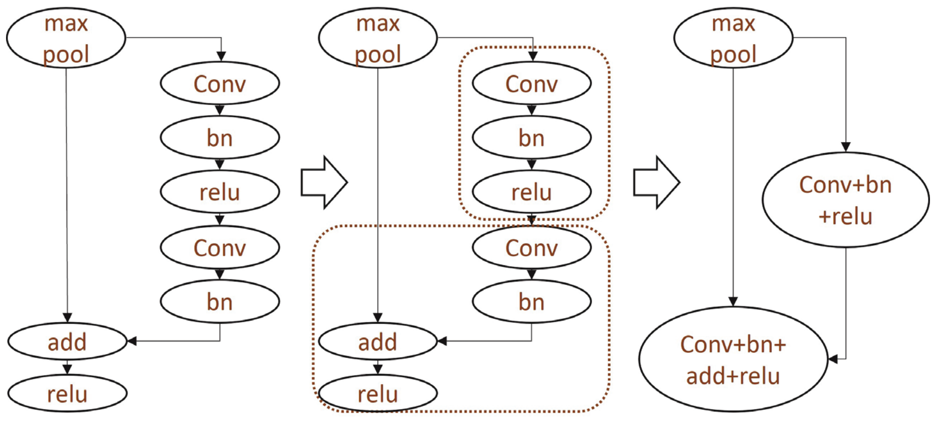


FIGURE 2. A kernel search example on a subgraph of ResNet18 model. The found kernels are maxpool, Conv+bn+relu, Conv+bn+add+relu.

WE PROPOSE *nn-METER*… TO PREDICT THE DNN INFERENCE LATENCY ON DIVERSE EDGE DEVICES

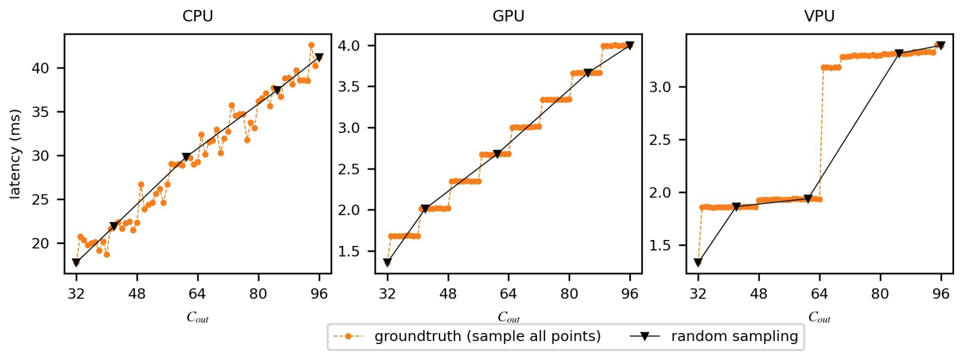


FIGURE 3. Latency of Conv+bn+relu with different output channel number C_{out}. The groundtruth exhibits a staircase pattern on GPU and VPU. Random sampling misses many hardware-crucial data. (HW=112, C_{in} =32, K=3, S=1)

(Depth-first search) rule matching algorithm is designed to search for the maximum fusion unit (i.e., kernel) in a model.

Test Case Design. Our test case design is driven by two features of a NN model, which can impact the fusion rules on target devices, i.e., *operator type* and *operator connection*. Operator type can impact fusion rules because the fusion of different operators requires different implementation efforts. For example, the code of injective operators can be easily connected (and thus fused) to the code of other operators compared to that of non-injective operators. Operator connection also impacts fusion rules. This is because improper fusion may not only cause additional time cost, but also cyclic operator dependency. Although the model graphs are arbitrary, they are all composed of three basic operator connection types, i.e., single in/outbound, multi-outbound and multi-inbound, which are the targets of the fusion rules.

Based on the analysis above, for operator type, our test cases include the single in/outbound connection permutation of every two possible operators, to detect whether they can be fused. Then, the fusible operators are selected to compose multi-in/outbound connections to detect the rules for different connections.

The inference latency difference of connected and separated operators is used as the metric to judge whether fusion happens. That is, for a single in/outbound connection ($op1$, $op2$), if the time of operators follows the formula $T_{op1} + T_{op2} - T_{(op1,op2)} > \alpha^{*}\min(T_{op1}, T_{op2})$, they are regarded as being *fused* as $op1+op2$. In the formula, T_{op1} and $T_{(op1,op2)}$ are the measured time of $op1$ and ($op1$, $op2$) connection respectively. α is the empirical coefficient as a threshold.

Find All Kernels of a Model. With the detected fusion rules, for a model graph, nn-Meter recursively applies the rules to the graph to find all the constituent kernels (i.e., fused operators). For example, with matching the fusion rules on mobile GPU, nn-Meter divides a ResNet18 subgraph into three different kernels in Figure 2.

LATENCY PREDICTOR

Challenges

By applying the kernel detection to the target model, we get a set of kernels. The next step is to build latency predictors for these kernels. However, it is non-trivial to build accurate kernel predictors. The kernels show non-linearity between latency and prediction features (shown in Figure 3). Moreover, the multiple configurable dimensions of kernels lead to a huge possible sampling space for the latency prediction. For instance, Conv kernels usually have a 6-dimension of configuration parameters: input height H, input width W, kernel size K, stride S, input channel number C_{in}, and output channel C_{out}. The size of the sample space is the multiplication of the size of every dimension and can easily reach billions. Sampling the whole space to get labeled training data is infeasible. Thus, how to do efficient sampling while ensuing high prediction accuracy remains a big challenge.

To reduce the data sampling cost, nn-Meter uses an adaptive data sampling algorithm that leverages both the model design and hardware latency characteristics.

It firstly prunes the kernel configurations that are rarely considered in DNN models. Then, an iterative sampling process is executed to automatically detect the most beneficial configurations to sample, instead of random selection. Finally, we build machine-learning regressors to learn the non-linearity with the sampled data.

Adaptive Data Sampling. We now describe the main steps of the adaptive data sampling algorithm in Figure 4. First, to generate sufficient configurations that are likely to be considered in NN design, we sample by a prior possibility distribution, which is calculated with kernel configurations in existing CNN models. The distribution describes the boundary and the possibility of each data (i.e., kernel configuration) to sample. Through sampling from the distribution, we can prune lots of rarely considered configurations.

Then, we run an iterative process to sample more data around inaccurate prediction data. Since large errors usually indicate that prediction model requires more information around them, we treat them as the hardware-crucial data and perform more fine-grained sampling. At each iteration, we first collect all the available data to update the machine learning predictor. We adapt the Random Forests Regression as the predictor model, which can learn the non-linearity. Then, we use the predictor to evaluate each data in the test set and pick out those with large errors. We further conduct fine-grained sampling around them. Specifically, we leverage our observation in Figure 3 to sample more data in the channel number C dimension. For each data, we fix all the other dimensions except the channel number C. We randomly sample M data from $[0.4 \times C, 1.2 \times C]$. The iterative process continues until the predictor accuracy meets the user's requirements.

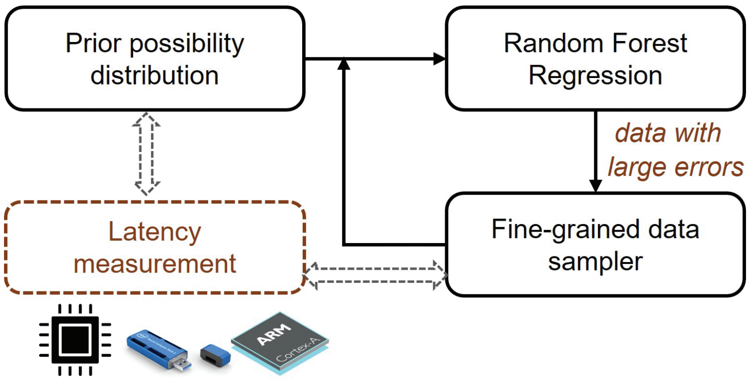


FIGURE 4. The adaptive data sampling algorithm.

Finally, we collect all the sampled data to build the predictors for kernels. For a given model, we sum up all the kernels' predicted latencies as the model latency.

EVALUATION

Benchmark dataset collection. To evaluate the effectiveness of nn-Meter on an arbitrary DNN model, we need a representative dataset that covers a large prediction scope. First, we collect 12 state-of-the-art CNN models on the ImageNet2012. For each model, we generate 2,000 variants by re-sampling the output channel number and kernel size for each layer. Besides, we add 2,000 models with the highest test accuracy on CIFAR10 from the NASBench201, where each model has a different set of edge connections. In total, our dataset contains 26,000 models.

End-to-end prediction results. We evaluate nn-Meter on the benchmark dataset for 4 types of devices in Table 1. We predict the latency of 26,000 models on each evaluated device. We report the ±10% accuracy [7], that are the percentage of models with predicted latency within the corresponding error bound relative to the measured latency. Remarkably, we achieve 99.0%, 99.0% and 99.1% prediction accuracy on the CPU, Adreno630 and Adreno640 GPU, respectively. On the Intel VPU, we can predict 83.4% models within the ±10% error boundary. With detailed manual analysis, we found that VPU performs ad-hoc optimizations that merges the computation of Conv+bn+relu and the next maxpool layer in VGG models. Fortunately, these ad-hoc optimizations are rare, and their impact on prediction accuracy is limited.

Comparison with baselines on unseen models. In real-world scenarios, a usable predictor must be able to predict unseen models (i.e., a new model). nn-Meter requires no model-level data for building the predictors and can make predictions on models it has not seen before. To demonstrate it, we implement 3 baselines for comparison: (1) FLOPs, (2) FLOPs+MAC, (3) BRP-NAS. Baselines (1) and (2) are the widely used latency predictors. Baseline (3) is the latency predictor in BRP-NAS, one of the state-of-the-art model-graph based prediction by GCN on the NASBench201 dataset. Since the three baselines require model-level information, we design a k-fold cross-validation experiment for evaluation.

Figure 5 shows the prediction accuracy achieved by different predictors. Compared with the baselines, nn-Meter is the only approach that consistently achieves accurate predictions on various devices. None of the baselines can achieve comparable performance for unseen models on any device. Specifically, on average, nn-Meter achieves 89.2% accuracy, significantly better than FLOPs (22.1%), FLOPs+MAC (17.1%), and BRP-NAS (8.5%) on the three devices.

TABLE 1. End-to-end latency prediction for 26,000 models on mobile CPU, GPU and Intel VPU.

Device	Processor	Framework	Accuracy
Pixel 4	CortexA76 CPU	TFLite v2.1	99.0%
Pixel 3XL	Adreno630 GPU	TFLite v2.1	99.0%
Mi9	Adreno640 GPU	TFLite v2.1	99.1%
Intel NCS2	Myriad VPU	Openvino 2019R2	83.4%

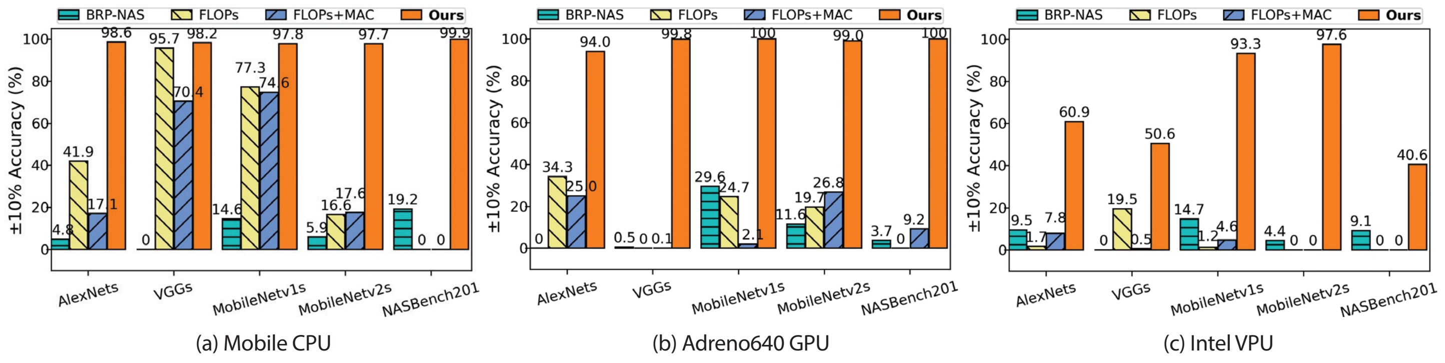


FIGURE 5. Compared to the baselines, nn-Meter achieves much higher accuracy on unseen models.

CONCLUSION

We propose nn-Meter, a kernel-based prediction system that accurately predicts the latency of DNN models on diverse edge devices. nn-Meter introduces kernel detection that captures the various operator-fusion behaviors. By sampling the most beneficial data, nn-Meter efficiently builds latency predictors for kernels. We demonstrate the effectiveness of nn-Meter with experiments on a large dataset and four types of edge devices.

While we have obtained promising results of nn-Meter on the three platforms, it requires joint efforts across the community to apply nn-Meter onto many other types of edge devices. To this end, we open-source[1] our code for other researchers and developers to build latency predictors for their own devices. Collectively, we expect that the community can work together to realize accurate latency prediction of DNN models for a variety of edge devices. ■

Li Lyna Zhang is a senior researcher at Microsoft Research. Her research interests are edge AI computing, hardware aware AutoML and model compression. She received her Ph.D. (2018) and B.S. (2013) at University of Science and Technology of China.

Shihao Han is a third-year student at Microsoft Research, Rose-Hulman Institute of Technology, where he is pursuing a degree in Electrical Engineering. His research interests lie primarily in efficient edge AI.

Jianyu Wei is a first-year Master's student at University of Science and Technology of China. He received his B.S. at University of Science and Technology of China in 2021.

Ningxin Zheng is a research software developer at Microsoft Research. His research interests are AI systems, cloud computing, and datacenter. He received his M.S. from Shanghai Jiao Tong University in 2020.

Ting Cao is a senior research manager in Microsoft Research. Her research interests include deep learning systems, Hardware/Software co-design, high-level language implementation, and energy efficient hardware design. She received her PhD from the Australian National University. Before joining MSRA, she worked in 2012 Labs, Huawei Technologies.

Yuqing Yang is principal research SDE manager at Microsoft Research. His research focuses on efficient AI systems, such as accelerating deep learning training and inference, and creating specialized clusters for deep learning workloads. He received his Ph.D. ('11) and B.S. ('06) from Fudan University, Shanghai, China.

Yunxin Liu is a Guoqiang professor at the Institute for AI Industry Research (AIR), Tsinghua University. His research interests are mobile computing and edge computing. He received his Ph.D. from Shanghai Jiao Tong University in 2011, M.S. from Tsinghua University in 2001, and B.S. from University of Science and Technology of China in 1998.

[1] https://github.com/microsoft/nn-Meter

REFERENCES

[1] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. *The World Wide Web Conference (WWW).*

[2] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR).*

[3] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. *European Conference on Computer Vision (ECCV).*

[4] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR).*

[5] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 10734–10742.

[6] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Tim Kwang-Ting Cheng Xin Yang, and Jian Sun. 2019. MetaPruning: Meta learning for automatic neural architecture channel pruning. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).*

[7] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. 2020. BRP-NAS: Prediction-based NAS using GCNs. *Advances in Neural Information Processing Systems (Neurips).*

[8] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR).*