AutoScale: Energy Efficiency Optimization for Stochastic Edge Inference Using Reinforcement Learning

Young Geun Kim* and Carole-Jean Wu*†
Arizona State University* Facebook AI†
{younggeun.kim, carole-jean.wu}@asu.edu

***Abstract*—Deep learning inference is increasingly run at the edge. As the programming and system stack support becomes mature, it enables acceleration opportunities in a mobile system, where the system performance envelope is *scaled up* with a plethora of programmable co-processors. Thus, intelligent services designed for mobile users can choose between running inference on the CPU or any of the co-processors in the mobile system, and exploiting connected systems such as the cloud or a nearby, locally connected mobile system. By doing so, these services can *scale out* the performance and increase the energy efficiency of edge mobile systems. This gives rise to a new challenge—deciding *when* inference should run *where*. Such execution scaling decision becomes more complicated with the stochastic nature of mobile-cloud execution environment, where signal strength variation in the wireless networks and resource interference can affect real-time inference performance and system energy efficiency. To enable energy efficient deep learning inference at the edge, this paper proposes *AutoScale*, an adaptive and lightweight execution scaling engine built on the custom-designed reinforcement learning algorithm. It continuously learns and selects the most energy efficient inference execution target by considering characteristics of neural networks and available systems in the collaborative cloud-edge execution environment while adapting to stochastic runtime variance. Real system implementation and evaluation, considering realistic execution scenarios, demonstrate an average of 9.8x and 1.6x energy efficiency improvement over the baseline mobile CPU and cloud offloading, respectively, while meeting the real-time performance and accuracy requirements.**

I. Introduction

It is expected that there will be more than 7 billion mobile device users and 900 million wearable device users in 2021 [99], [100], with products including smartphones, smartwatches, and wearable virtual- or augmented-reality devices. To improve the mobile user experience, various intelligent services such as virtual assistant [3], [5], face/image recognition [36], and language translation [38] have been introduced in recent years. Many companies, including Amazon, Facebook, Google, and Microsoft, are using sophisticated machine learning models, especially deep neural networks (DNNs), as the key component of their intelligent services [3], [38], [77], [111].

Traditionally, due to the compute- and memory-intensive nature of the DNN workloads [7], [18], [44], both training and inference have been executed on the cloud [25], [53], while the mobile devices only acted as user-end sensors, user interfaces, or both. More recently, with the advancements in powerful mobile systems-on-a-chip (SoCs) [41], [50], [107], there has been increasing push to execute DNN inference on the edge mobile devices [10], [25], [42], [53], [55], [65], [106], [107], [111], [121]. This is because executing inference on the edge devices can improve the response time of services by removing data transmission overhead. However, executing inference on the edge mobile devices also increases the energy consumption of mobile SoCs [53]. Since these devices are energy constrained [60], it is necessary to optimize energy efficiency of the DNN inference, satisfying quality-of-service (QoS) requirements.

To address the performance and energy efficiency challenges, mobile devices are employing more and more accelerators and co-processors, such as graphics processing units (GPUs), digital signal processors (DSPs), and neural processing units (NPUs) [12], [51], *scaling up* the overall system performance. Furthermore, the mobile system stack support for DNNs has become more mature, allowing DNN inference to benefit from the computation and energy efficiency advantages provided by the co-processors. For example, deep learning compiler and programming stacks, such as TVM [12], SNPE [90], and Android NNAPI [4], [51], enable inference execution on diverse hardware back-ends including the co-processors.

These recent advancements give rise to a *new* challenge—deciding *when* inference should run *where*. Intelligent services that run on a mobile device can run inference on the CPU or any available co-processor in the device, or exploit connected systems such as the cloud or a nearby, locally-connected system [6] that is more powerful than the device itself. By doing so, the services can *scale out* the performance and increase the energy efficiency of edge inference. For example, many personalized health and entertainment applications operate in a collaborative execution environment composed of smartwatches, smartphones, and the cloud [29], [46], [85], [109]. Similarly, virtual- and augmented-reality systems comprise wearable electronics, smartphones (as the staging device), and the cloud [35], [37], [78], [84]. However, deciding which execution target to exploit is challenging for any intelligent services, since the energy efficiency of each execution target significantly varies with various features, such as neural network (NN) characteristics and/or edge-cloud system profiles. The extremely fragmented mobile SoCs make this decision even more difficult, as there are myriad hardware targets with different profiles [111] to choose from.

To determine the optimal execution scaling decision, state-of-the-art approaches, such as [25], [42], [53], [106], [107], [121], proposed to rely on predictive models. However these approaches did not consider stochastic runtime variance, such as interference from co-running tasks and network signal

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00090

strength variation, which have a large impact on energy efficiency [34]. In a real use case, there can be several applications simultaneously running along with the DNN inference [57], [67], [98], because recent mobile devices support multi-tasking features [98], such as screen sharing between applications. In addition, signal strength variations in wireless networks can affect the performance and energy efficiency of cloud inference, as the data transmission latency and energy increase exponentially at weak signal strength [61] which accounts for 43% of data transmission [19]. Therefore, without considering such stochastic variance, one would not be able to choose the optimal execution scaling decision for DNN inference.

This paper proposes an adaptive and light-weight execution scaling engine, called *AutoScale*, to make accurate scaling decisions for the optimal execution target of edge DNN inference in the presence of stochastic variance. Since the optimal execution target varies with the NN characteristics, underlying execution platforms, and stochastic runtime variance, it is infeasible to enumerate the massive design space exhaustively. Therefore, AutoScale leverages a reinforcement learning technique for continuous learning, that captures and adapts to the stochastic environmental variance [20], [79], [83], [97]. AutoScale observes the NN characteristics, such as layer composition, and the system information, such as interference intensity and network stability. It then selects an execution target which is expected to maximize the DNN inference energy efficiency while satisfying the performance and accuracy targets. The result of the selection is measured from the system and fed back to AutoScale, allowing it to continuously learn and predict the optimal execution target. We demonstrate AutoScale with real system-based experiments that show 9.8x and 1.6x improved average energy efficiency, compared to the baseline settings of mobile CPU and cloud offloading, respectively, satisfying both the QoS and accuracy constraints with 97.9% prediction accuracy.

This paper makes the following key contributions:

- This paper provides an in-depth characterization of DNN inference execution on mobile and edge-cloud systems. The characterization results show that the optimal execution scaling decision significantly varies with the NN characteristics and the stochastic nature of mobile execution (Section III).
- This paper proposes an intelligent execution scaling engine, called AutoScale, that accurately selects the optimal execution target of edge inference in the presence of stochastic variance (Section IV).
- To demonstrate the feasibility and practicality of the proposed execution scaling engine, we implement and evaluate AutoScale with a variety of on-device inference use cases in an edge-cloud execution environment using real systems and devices, allowing AutoScale to be adopted immediately[1] (Section VI).

[1]https://github.com/mocha-research/AutoScale

II. Background

A. Deep Neural Network

DNNs connect numerous functional layers to extract features from inputs at multiple levels of abstraction [54], [66]. Each layer comprises multiple processing elements (neurons), which are applied with the same function to process different parts of an input. Depending on what function is applied, the layers can be classified into various types [18]. These layers and their execution characteristics are important since they can affect the decision made by AutoScale. We provide brief descriptions for each type below.

A *convolutional layer* (**CONV**) performs a two-dimensional convolution to extract a set of feature maps from an input. To selectively activate meaningful features, it applies an activation function such as sigmoid or rectified linear to the obtained feature maps. Typically, this layer is compute-intensive due to the convolutions.

A *fully-connected layer* (**FC**) computes the weighted sum of the inputs and then applies the activation function to the sum. This layer is one of the most compute- and memory-intensive layers in DNNs [18], [53], [55], since its neurons are connected exhaustively to all neurons in the previous layer.

A *recurrent layer* (**RC**) uses the output of a given step in a sequence as an input in the next step. At each step, this layer also computes the weighted sum of the inputs. This layer is even more compute- and memory-intensive than an FC layer, since its neurons can be connected to those in the previous, current, and next layers.

Other common layers include the following: A *pooling layer* (**POOL**) applies a sub-sampling function, such as max or average, to regions of the input feature maps; a *normalization layer* normalizes features across spatially grouped feature maps; a *softmax layer* yields a probability distribution over the possible classification categories; an *argmax layer* chooses the class with the highest probability; and a *dropout layer* randomly ignores neurons during training and allows them to pass through during inference. These layers are typically less compute- and memory-intensive than CONV, FC, and RC layers, so that they usually have little impact on performance and energy efficiency of DNN inference.

DNNs can have various layer compositions. For example, computer vision NNs (e.g., Inception, MobileNet, and ResNet) mainly comprise CONV, POOL, and FC layers. On the other hand, language processing NNs (e.g., BERT) mainly consist of RC layers, such as long short-term memory (LSTM) and attention. Since each layer has unique characteristics due to different compute- and memory-intensities, to optimize inference execution for DNNs, it is important to consider the layer compositions.

B. DNN Inference Execution at the Edge

Fig. 1 depicts the general structure of the system stack for machine learning inference execution at the edge. At the front-end, DNNs are implemented with various frameworks [9], [82], [89], [105], whereas the middleware allows the deployment of DNN inference on diverse hardware back-ends. The frameworks

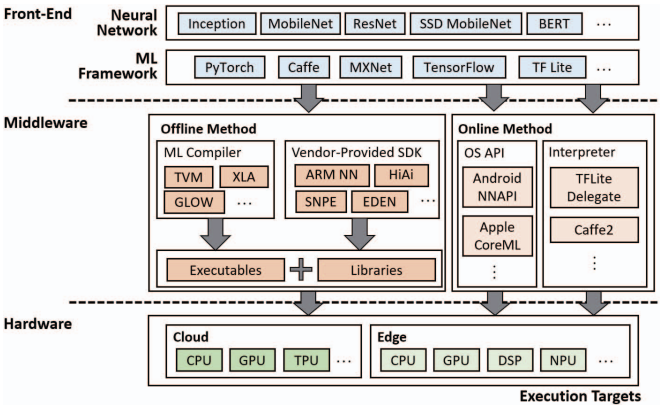


Fig. 1. System stack for DNN inference.

and middleware also enable efficient inference at the edge—various NN optimizations, such as quantization [15], [30], [52], [62], [65], [111], [119], weight compression [43], [68], and graph pruning [112], [118] can be employed before the DNNs are deployed. Among them, quantization is one of the most widely used ones for the edge execution, since it reduces both compute- and memory-intensities of the inference by shrinking 32-bit floating-point (FP32) values to, for example, 16-bit floating-point (FP16) or 8-bit integer (INT8). Since the middleware does not select a specific hardware target for DNN inference execution, intelligent services should choose one among the possible hardware targets. However, this decision process is challenging, as the energy efficiency of each target can considerably vary with various features.

C. Real-Time Inference Quality of Experience

The quality of user experience is a crucial metric for mobile optimization. For real-time inference, the quality of experience (QoE) is the product of system energy efficiency, inference latency, and inference accuracy. To improve energy efficiency of mobile devices, a number of energy management techniques can be used [60]. Unfortunately, the techniques often sacrifice performance (i.e., latency) for energy efficiency, degrading QoE of real-time inference.

Inference latency is an important QoE factor, because if the latency of a service exceeds the human-acceptable limit, users will abandon the service [98], [122]. However, a single-minded pursuit of performance is undesirable due to energy constrained nature of mobile devices. Hence, there is a need to provide just enough performance to meet the QoS expectations of users with minimal energy consumption. These expectations can be defined as a certain latency (e.g., 33.3 ms for 30 FPS video frame rate [22], [122], or 50 ms for interactive applications [23], [74]), below which most users cannot perceive any difference.

Various NN optimizations can improve both the latency and energy efficiency of inference, but they often sacrifice accuracy. Since human-level accuracy is a primary requirement for user satisfaction [7], [18], [55], it is also important to keep the inference accuracy above the quality expectation of users.

In summary, to maximize the quality of user experience for real-time inference, it is crucial to maximize the system-wide energy efficiency while satisfying the human acceptable latency and accuracy expectations.

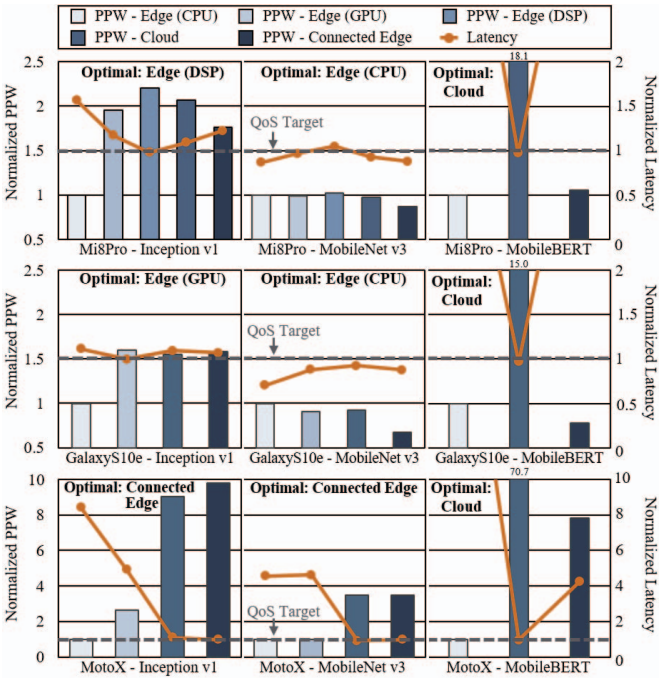


Fig. 2. The optimal execution target depends on NN characteristics and the edge-cloud system profiles. Performance per watt (PPW) is normalized to Edge (CPU) and latency is normalized to the QoS target.

III. Motivation

This section presents system characterization results for realistic DNN inference scenarios on actual mobile and edge-cloud systems. We examine the design space covering three important axes—latency, accuracy, and energy efficiency (performance per watt).

For mobile inference, we select three smartphones—Xiaomi Mi8Pro, Samsung Galaxy S10e, and Motorola Moto X Force—to respectively represent high-end mobile systems with GPU and DSP co-processors, high-end mobile systems with GPU but without DSP, and mid-end mobile systems[2].

We emulate edge-cloud inference execution using the three smartphones and a server-class Intel Xeon processor, hosting an NVIDIA P100 GPU. For a locally connected mobile device, we use a tablet, Samsung Galaxy Tab S6; note we connect the smartphones to the tablet over a Wi-Fi-based peer-to-peer network (Wi-Fi Direct). Detailed specifications of our mobile and edge-cloud setup appear in Section V.

A. Varying Optimal DNN Execution Target

- *Optimal edge-cloud execution depends on the NN characteristics and edge-cloud system profiles.*

Fig. 2 shows the energy efficiency and latency of three common mobile inference use cases over the three mobile

[2]We use high-end mobile systems with and without an NN-specialized accelerator (i.e., a DSP) to examine the performance scale-up from off-the-shelf mobile systems. In addition, we select the Moto X Force to represent mid-end mobile systems with much wider market coverage [111] (for details, see Section V).

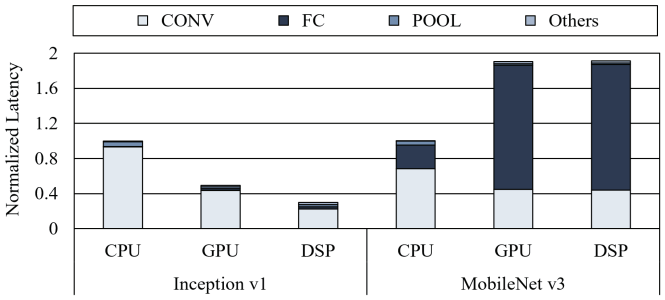
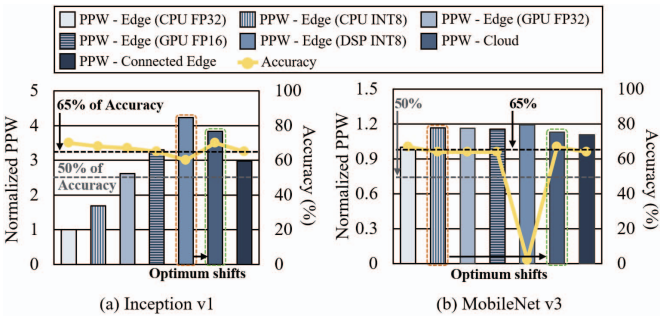


Fig. 3. Each NN layer exhibits different latency on different mobile processors. The optimal execution target thus depends on layer compositions. Note that latency is normalized to that of CPU.



(a) Inception v1 (b) MobileNet v3

Fig. 4. Depending on the inference accuracy target, optimal edge-cloud execution also shifts. Note that PPW is normalized to Edge (CPU FP32).

devices and the edge-cloud setup. The x-axis represents the mobile system running three representative NNs.

For the high-end systems (i.e., Mi8Pro and Galaxy S10e), the optimal edge-cloud execution depends on the NN characteristics. For example, in the case of light NNs such as Inception v1 [103] and MobileNet v3 [47], edge inference is more efficient than cloud inference. This is because off-the-shelf mobile SoCs deliver enough performance to satisfy the QoS target of the light NNs. On the other hand, cloud inference is more efficient than edge inference for the heavy NNs, such as MobileBERT [101], since the performance of the mobile SoCs is insufficient. In this case, the performance gain of cloud execution (reduced computation time and energy) outweighs the loss (increased data transmission time and energy).

For the mid-end system (i.e., Moto X Force), however, scaling out to the connected systems is always beneficial, as the performance of the SoC in this system is not enough even for the light NNs. For the light NNs, scaling out to a locally connected device can be an option, since 1) the higher-end device (i.e., tablet) can satisfy the QoS constraint of the light NNs, and 2) data transmission overhead between the locally connected edge devices is usually smaller than that between edge-cloud. On the other hand, in case of heavy NNs, there is no option other than scaling out to the cloud.

- *The optimal execution target depends on layer compositions.*

Another important observation in edge inference execution is that the optimal execution target can vary with the layer compositions of the NNs. Fig. 3 shows the cumulative latency of different layers in two NNs[3], running on different processors in the Mi8Pro. The compute- and memory-intensive FC layers exhibit much longer latency on co-processors, whereas other layers exhibit longer latency on CPUs. NNs that have more FC layers (e.g., MobileNet v3) thus run more efficiently on CPUs, while others (e.g., Inception v1) run more efficiently on co-processors. This result also implies that the co-processors do not always outweigh the CPUs, so that carefully choosing one considering layer compositions is crucial for energy efficiency.

- *The optimal edge-cloud execution target varies with the inference quality requirement.*

[3]MobileBERT was not used for this experiment, since the inference execution of MobileBERT on co-processors is not supported by any middleware yet.

Fig. 4 shows the energy efficiency (PPW) and the DNN inference accuracy on different execution targets, where the inference quality (i.e., accuracy) of each NN highly depends on the execution target. Note that the accuracy for each processor is measured in our edge-cloud systems using the ImageNet validation set [17]. If the accuracy requirement is 50%, the optimal target may be DSP INT8 and CPU INT8 for Inception v1 and MobileNet v3, respectively; these targets show the highest energy efficiency while satisfying the QoS constraint. If the accuracy requirement is 65%, however, the optimal target should be shifted to the cloud to satisfy the accuracy requirement.

B. Impact of Runtime Variance on Inference Execution

In a realistic environment, there can be on-device interference from co-running applications [57], [67], [98]. In addition, the network signal strength can vary considerably as edge device users move. In fact, users undergo significant signal strength variations in daily life (43% of data is transmitted under weak signal strength [19]).

- *On-device interference and varying network stability change the optimal edge-cloud execution.*

Fig. 5 shows the normalized energy efficiency (PPW) and latency of DNN (i.e., MobileNet v3) inference when CPU-intensive or memory-intensive synthetic applications are co-running. When a CPU-intensive application is co-running, the energy efficiency of the inference execution on CPU is significantly degraded because of competition for CPU resources and frequent thermal throttling due to high CPU utilization [59]. In this case, the optimal execution target shifts from the CPU to the GPU. On the other hand, when a memory-intensive application is co-running, the energy efficiency of all on-device processors (including the CPU, GPU, and DSP) is degraded since the inference execution is competing with other applications for the memory. The optimal target therefore moves from the edge to the cloud.

Fig. 6 shows the normalized energy efficiency (PPW) and latency of DNN (i.e., ResNet 50 [45]) inference when the wireless network signal strength varies. When the signal strength weakens, inference execution on connected systems becomes less energy efficient, since 1) the data transmission time exponentially increases with decreased data rate [19], [61], and 2) the network interface consumes more power to transmit

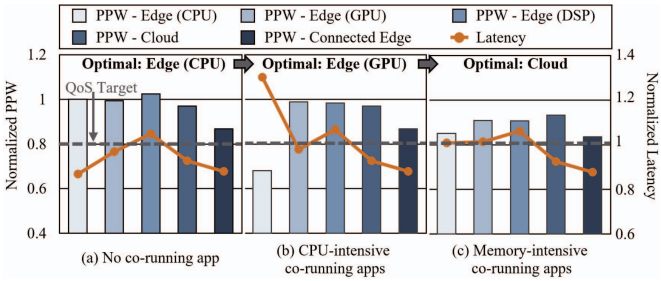


Fig. 5. In the presence of on-device interference, the optimal edge-cloud execution target varies. Note that PPW is normalized to Edge (CPU) with no co-running app and latency is normalized to the QoS target.

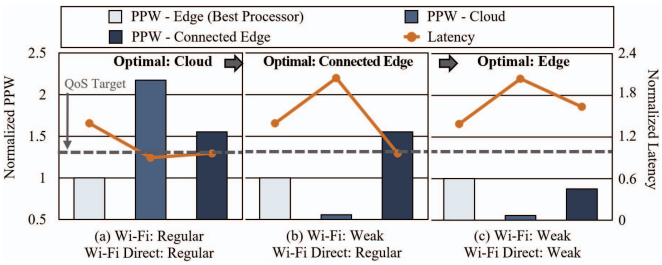


Fig. 6. When signal strength varies, the optimal edge-cloud execution target also varies. Note that PPW is normalized to Edge (Best Processor) and latency is normalized to the QoS target.

data with stronger signals. If only the Wi-Fi signal strength weakens, the locally connected edge device can still serve as an optimal execution target. However, if the Wi-Fi Direct signal strength also weakens, the optimal target shifts to the edge.

C. Inefficiency of Prediction-based Approaches

Energy optimization of mobile DNN inference can be formulated as the problem of choosing the optimal execution target in the presence of stochastic runtime variance, which optimizes energy efficiency while satisfying the QoS and accuracy constraints. One possible solution for this kind of problems is to evaluate all execution targets using a prediction model. Unfortunately, owing to the massive design space and the nonlinear relationship across NN characteristics and runtime variance, it is difficult to simply build an accurate prediction model. An inaccurate prediction can result in selection of a sub-optimal execution target.

- *Exhaustively enumerating the massive design space is infeasible. Simple prediction-based approaches are insufficient, leaving significant room for energy efficiency improvement.*

To shed light on the inefficiency of existing prediction-based approaches, we compare three types of prediction-based approaches with the baseline (Edge (CPU)) and oracular design (Opt): 1) regression-based approaches, 2) classification-based approaches, and 3) Bayesian optimization-based approach [32], [39], [92]. For each type, we use methods that are widely adopted by existing works in this domain [10], [25], [42], [53], [121]. For the regression-based approaches, we use linear regression (LR) [96] and support vector regression (SVR) [21]. For the classification-based approaches, we select support vector machine (SVM) [102] and k nearest neighbor

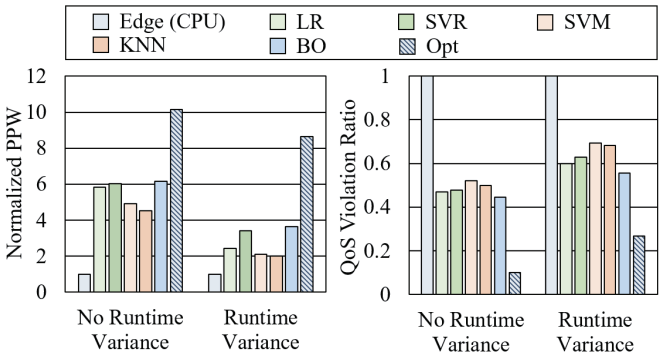


Fig. 7. There is a significant gap between Opt and existing prediction-based approaches, as they fail to accurately predict the optimal execution target in the presence of runtime variance.

(KNN) [114]. The objective of Bayesian optimization (BO) is set to find the execution target that maximizes energy efficiency while satisfying the QoS constraint. We employ the Gaussian process as the surrogate model and expected improvement as the acquisition function. Using BO, we obtain the energy efficiency and latency estimation functions and use them to predict the optimal target at runtime.

Fig. 7 shows the energy efficiency (PPW) and QoS violation ratio of prediction-based approaches normalized to those of Edge (CPU). Although these approaches improve energy efficiency compared to the baseline, there is a significant gap between the approaches and Opt, as they fail to accurately select the optimal execution target.

When there is no runtime variance, the mean absolute percentage error (MAPE) is 13.6% for LR and 10.8% for SVR. But when stochastic runtime variance is present, the MAPE for LR and SVR is 24.6% and 21.1%, respectively. Because of the inaccurate energy and latency predictions, these approaches fail to run DNN inference on the optimal execution target, degrading energy efficiency and violating the QoS constraint. On the other hand, the mis-classification ratios of SVM and KNN are 12.7% and 14.3%, respectively, when runtime variance is present. Although the two values do not seem to be large, these approaches degrade energy efficiency much more than regression-based approaches. This is because they make the wrong decision regardless of the absolute energy and latency magnitudes. For example, even though on-device inference is much more efficient than cloud inference when the signal strength is weak, cloud inference can be selected as the execution target. The BO-based approach also fails to accurately capture the impact of runtime variance—its MAPE with and without runtime variance is 15.7% and 9.2%, respectively. Hence, they also degrade the energy efficiency and latency, leaving significant room for energy efficiency improvement.

These results call for a novel scheduler design that can accurately select the optimal DNN inference execution target while adapting to stochastic runtime variance. In the next section, we present our proposed AutoScale design, which employs reinforcement learning to self-learn the optimal execution target in the presence of runtime variance.

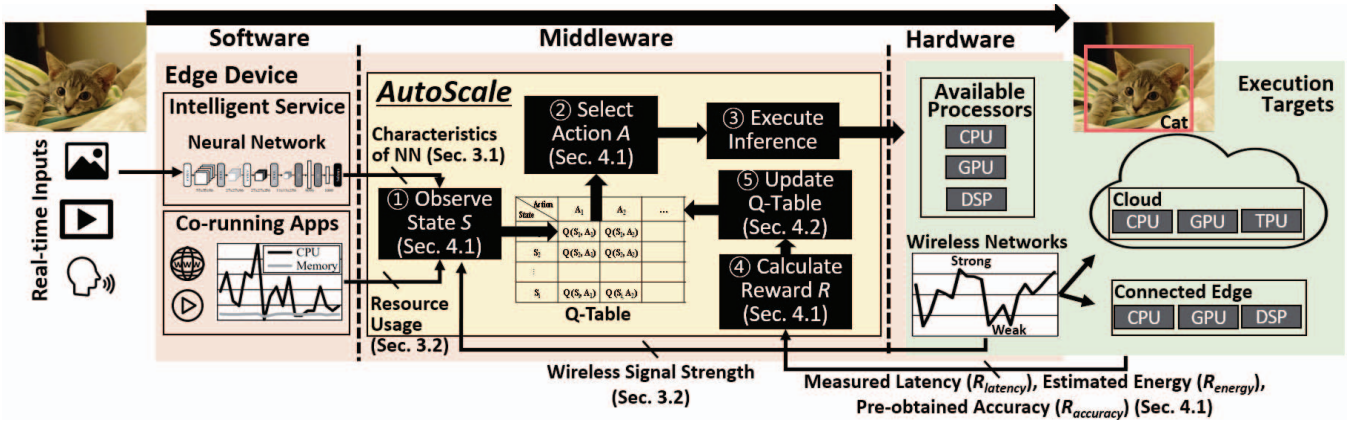


Fig. 8. AutoScale design overview.

IV. AutoScale

Fig. 8 shows the design overview of AutoScale in the context of the mobile and edge-cloud DNN inference execution. For each inference, AutoScale identifies the current execution state (①), including NN characteristics and runtime variance. For the observed state, AutoScale selects an action (i.e., execution target ②), that is expected to maximize energy efficiency satisfying QoS and inference quality requirements. This selection is based on a lookup table (i.e., Q-table) that contains the accumulated rewards of previous selections. AutoScale then executes the DNN inference on the target[4] defined by the selected action (③) while observing the result (i.e., energy, latency, and inference accuracy). Based on that result, AutoScale calculates the reward (④), which indicates how much the selected action improves energy efficiency and satisfies the QoS and accuracy targets. Finally, AutoScale updates the Q-table with the calculated reward (⑤).

AutoScale employs reinforcement learning (RL) as an adaptive prediction mechanism. Generally, an RL agent learns a policy to select the best action for a given state with accumulated rewards [86]. In the context of mobile and edge-cloud inference execution, AutoScale learns how to select the optimal inference execution target for a given NN in the presence of runtime variance based on the accumulated energy, latency, and accuracy results of selections. To solve system optimization using RL, ther are three important design requirements for mobile use.

High Prediction Accuracy: The success of RL depends on how much the predicted execution target is close to the optimal one. For accurate predictions, it is crucial to model the core components—*state*, *action*, and *reward*—in a realistic environment. We define these components in accordance with our observations of a realistic edge inference execution environment (Section IV-A).

In addition to the core components, avoiding local optima is also important. It is deeply related to a classical RL problem: exploitation versus exploration [27], [64]. If an RL agent always exploits an action with the temporary highest reward, it can get stuck in local optima. On the other hand, if it keeps exploring all possible actions, convergence may get slower. To solve this problem, we employ the epsilon-greedy algorithm, which is one of the widely adopted randomized greedy algorithms in this domain [75], [83], [86], for its effectiveness and simplicity (Section IV-B).

Minimal Training Overhead: In case of RL, training is continuously performed on-device. Reducing its overhead is therefore crucial, particularly for the energy-constrained edge domain. As we observed in Section III, although performance of execution targets vary across heterogeneous devices, they all exhibit a similar energy trend for each NN. An RL model trained in a device has this energy trend knowledge implicitly. Hence, we consider transferring a model trained on one device to other devices in order to expedite the convergence, reducing the training overhead. (detailed results appear in Section VI-C).

Low Latency Overhead: For the real-time inference execution on energy-constrained edge devices, latency overhead is another crucial factor. Among the various RL approaches [86], such as Q-learning [14], TD-learning [70], and deep RL [79], Q-learning has an advantage for low latency overhead, as it employs a lookup table to find the best action. We therefore use Q-learning for AutoScale.

A. AutoScale RL Design

In RL, there are three core components: state, action, and reward. This section defines the core components to formulate the optimization space for AutoScale.

State: Based on our observations in Section III, we identify states that are critical to edge inference; Table I summarizes the states.

As we explored in Section III-A, the optimal target depends on the NN layer compositions. However, identifying states for all layer types is undesirable, since the latency overhead (i.e., Q-table lookup time) increases. Hence, we identify states with layer types that are deeply correlated with the energy efficiency and performance of inference execution. We test the correlation between each layer type and energy/latency by calculating the squared correlation coefficient (ρ^2) [123]. We find CONV, FC,

[4]In this paper, we only consider offloading at model granularity, since model partitioning at layer granularity introduces additional context switching overhead (from transferring intermediate inputs to another execution target). Model partitioning at layer granularity is complementary to and can be applied on top of AutoScale [42], [53], [55].

TABLE I
STATE-RELATED FEATURES.

State		Description	Discrete Values
NN-related Features	S_{CONV}	# of CONV layers	Small (<30), medium (<50), large (<90), larger (>=90)
	S_{FC}	# of FC layers	Small (<10), large (>=10)
	S_{RC}	# of RC layers	Small (<10), large (>=10)
	S_{MAC}	# of MAC operations	Small (<1,000M), medium (<2,000M), large (>=2,000M)
Runtime Variance	S_{Co_CPU}	CPU utilization of co-running apps	None (0%), small (<25%), medium (<75%), large (<=100%)
	S_{Co_MEM}	Memory usage of co-running apps	None (0%), small (<25%), medium (<75%), large (<=100%)
	S_{RSSI_W}	RSSI of wireless local area network	Regular (>–80dBm), weak (<=–80dBm)
	S_{RSSI_P}	RSSI of peer-to-peer wireless network	Regular (>–80dBm), weak (<=–80dBm)

and RC layers are most correlated with energy/latency, because of their compute- and/or memory-intensive nature. Thus, we identify S_{CONV}, S_{FC}, and S_{RC} which represent the number of CONV, FC, and RC layers, respectively. We also identify S_{MAC} as the number of MAC operations to consider size of NNs.

As we explored in Section III-B, the edge inference efficiency depends highly on the CPU-intensity and memory-intensity of co-running applications. Hence, we use S_{Co_CPU} and S_{Co_MEM} which represent the CPU and memory usage of co-running applications, respectively. In addition, the inference execution efficiency on the connected systems depends highly on the wireless network signal strength. We therefore use S_{RSSI_W} and S_{RSSI_P} which stand for the RSSI of wireless local area network (e.g., Wi-Fi, LTE, and 5G) and RSSI of peer-to-peer wireless network (e.g., Bluetooth, Wi-Fi Direct, etc.), respectively.

When a feature has a continuous value, it is difficult to define the state in a discrete manner for the lookup table of Q-learning [14], [83]. To convert the continuous features into discrete values, we applied DBSCAN clustering algorithm to each feature [14]; DBSCAN determines the optimal number of clusters for the given data. The last column of Table I shows discrete values for each state.

To examine the importance of each state, we conducted a sensitivity test using ablation [71], [120]. We found removing any one state degrades accuracy by 32.1% on average. This means that all the states are essential to predict the optimal execution target.

Action: RL actions represent the adjustable control knobs of the system. For edge-cloud inference, we define actions as the available execution targets. For the edge inference execution, available processors in mobile SoCs, such as CPUs, GPUs, DSPs, and NPUs, are defined as the actions. On the other hand, for the cloud execution, server-class processors, such as CPUs, GPUs, and TPUs, are defined as the actions.

The set of actions can be augmented by considering other control knobs, such as dynamic voltage and frequency scaling (DVFS) as well as quantization. For example, as long as the QoS constraint is satisfied, it is possible to reduce the frequency of processors, saving energy. In addition, employing the quantization for each processor can reduce both compute and memory intensities of the inference execution, improving energy efficiency and performance.

Reward: In RL, a reward models the optimization objective of the system. To represent the three main optimization axes, we encode three rewards: $R_{latency}$, R_{energy}, and $R_{accuracy}$. $R_{latency}$ is the measured inference latency for a selected action (i.e., execution target for DNN inference). R_{energy} is the estimated energy consumption of the selected action, and $R_{accuracy}$ is pre-measured inference accuracy of the given NN on each execution target.

We estimate R_{energy} for edge execution as follows. When the CPU is selected as the action, R_{energy} is calculated using the utilization-based CPU power model [55], [116] as in (1), where E^i_{Core} is the power consumed by the ith core, t^f_{busy} and t_{idle} are the time spent in the busy state at frequency f and that in the idle state, respectively, and P^f_{busy} and P_{idle} are the power consumed during t^f_{busy} at f and that during t_{idle}, respectively.

$$\begin{aligned} R_{energy} &= \sum_i E^i_{Core}, \\ E_{Core} &= \sum_f (P^f_{busy} \times t^f_{busy}) + P_{idle} \times t_{idle} \end{aligned} \tag{1}$$

Similarly, if scaling out the inference to GPUs within the system is selected as the action, R_{energy} is calculated using the GPU power model [58] as in (2). Note that t^f_{busy} and t_{idle} for CPU/GPU are obtained from *procfs* and *sysfs* in the Linux kernel [57], while P^f_{busy} and P_{idle} for CPU/GPU are obtained by power measurement of CPU/GPU at each frequency in the busy state and idle state, respectively. They are then stored in a look-up table of AutoScale.

$$R_{energy} = \sum_f (P^f_{busy} \times t^f_{busy}) + P_{idle} \times t_{idle} \tag{2}$$

If the selected action is to scale out the inference using DSPs, R_{energy} is calculated as in (3), where P_{DSP} is a pre-measured DSP power consumption; we use the constant value, since P_{DSP} remains consistent over 100 runs of 10 NNs.

$$E_{DSP} = P_{DSP} \times R_{latency} \tag{3}$$

On the other hand, if scaling out the inference execution to connected systems is selected as the action, R_{energy} is calculated using the signal strength-based energy model [61] as in (4), where t_{TX} and t_{RX} are the latencies measured when transmitting the input and receiving the output, respectively, and P^S_{TX} and P^S_{RX} are power consumed by a wireless network interface during t_{TX} and t_{RX}, respectively, at signal strength S. Note that P^S_{TX} and P^S_{RX} for each network are obtained my measuring power consumption of wireless network interfaces at each signal strength while transmitting and receiving data, respectively.

$$\begin{aligned} R_{energy} = & P^S_{TX} \times t_{TX} \; + \; P^S_{RX} \times t_{RX} \\ & + \; P_{idle} \times (R_{latency} - t_{TX} - t_{RX}) \end{aligned} \tag{4}$$

Since the energy estimation is based on the measured latency its MAPE is 7.3%, low enough to identify the optimal action.

Algorithm 1 Training Q-Learning Model

Variable: *S, A*
  S is the state
  A is the action
Constants: γ, μ, ε
  γ is the learning rate
  μ is the discount factor
  ε is the exploration probability
Initialize *Q(S,A)* as random values
Repeat (whenever inference begins):
  Observe state and store in *S*
  if rand() < ε **then**
    Choose action *A* randomly
  else
    Choose action *A* with the largest *Q(S,A)*
  Run inference on a target defined by *A*
  (when inference ends)
  Measure $R_{latency}$, estimate R_{energy}, and obtain $R_{accuracy}$
  Calculate reward *R*
  Observe new state *S'*
  Choose action *A'* with the largest *Q(S',A')*
  $Q(S,A) \leftarrow Q(S,A) + \gamma[R + \mu Q(S',A') - Q(S,A)]$
  $S \leftarrow S'$

To ensure AutoScale selects an efficient execution target that maximizes energy efficiency while satisfying the QoS and accuracy constraints, the reward *R* is calculated as in (5), where α is the latency weight and β is the accuracy weight; we use 0.1 for both, but we can use higher weights if the inference workload requires higher performance and accuracy.

$$
\begin{aligned}
&if \;\; R_{accuracy} < Inference\;\; Quality\;\; Requirement,\\
&\qquad R = R_{accuracy} - 100\\
&else\\
&\qquad if \;\; R_{latency} < QoS\;\; Constraint,\\
&\qquad\qquad R = -R_{energy} + \alpha R_{latency} + \beta R_{accuracy}\\
&\qquad else\\
&\qquad\qquad R = -R_{energy} + \beta R_{accuracy}
\end{aligned}
\tag{5}
$$

If the selected action fails to satisfy the inference quality requirement, the reward is $R_{accuracy}$ − 100 (i.e., how much the accuracy is far from 100%) to avoid choosing that action for the next inference. Otherwise, the reward is calculated depending on whether the QoS constraint is satisfied or not. In (5), R_{energy} is multiplied by –1 to increase the reward for lower energy consumption.

B. AutoScale Implementation

As we previously discussed, AutoScale uses Q-learning to exploit its low runtime overhead. To deal with the exploitation versus exploration dilemma in RL, AutoScale also employs the epsilon-greedy algorithm, which chooses the action with the highest reward or a uniformly random action based on an exploration probability.

In Q-learning, the value function *Q(S,A)* takes state *S* and action *A* as parameters. It is a form of lookup table, called a Q-table. Algorithm 1 shows the detailed algorithm for training the Q-table for on-device DNN inference. At the beginning, it initializes the Q-table with random values. At runtime, the algorithm observes *S* for each DNN inference by checking the NN characteristics and runtime variance. For the given *S*, it evaluates a random value compared with ε[5]. If the random value is smaller than ε, the algorithm randomly chooses *A* for exploration. Otherwise, it chooses *A* with the largest *Q(S,A)*. The algorithm then runs the inference on a target defined by *A*. During the inference, the algorithm measures $R_{latency}$ and estimates R_{energy}, as explained in Section IV-A. In addition, it obtains $R_{accuracy}$ from the stored inference accuracy for the given NN on the selected execution target. Using these values, the algorithm calculates the reward *R* as in (5) of Section IV-A. Afterward, it observes the new state *S'* and chooses the corresponding *A'* with the largest *Q(S',A')*. It then updates the *Q(S,A)* based on the equation in Algorithm 1. In the equation, γ and μ are hyperparameters that represent the learning rate and discount factor, respectively. The learning rate indicates how much the new information overrides the old information. On the other hand, the discount factor gives more weight to the rewards in the near future. We set γ and μ based on a sensitivity test (Section V-C).

After the learning is complete (i.e., the largest *Q(S,A)* value for each state *S* is converged), the Q-table is used to select *A* which maximizes *Q(S,A)* for the observed *S*.

TABLE II
MOBILE DEVICE SPECIFICATION WITH THE PEAK SYSTEM POWER CONSUMPTION SHOWN IN THE PARENTHESIS.

Device	CPU	GPU	DSP
Mi8Pro	Cortex A75 - 2.8GHz w/ 23 V/F steps (5.5 W)	Adreno 630 - 0.7GHz w/ 7 V/F steps (2.8 W)	Hexagon 685 (1.8 W)
Galaxy S10e	Mongoose - 2.7GHz w/ 21 V/F steps (5.6 W)	Mali-G76 - 0.7GHz w/ 9 V/F steps (2.4 W)	-
Moto X Force	Cortex A57 - 1.9GHz w/ 15 V/F steps (3.6 W)	Adreno 430 - 0.6GHz w/ 6 V/F steps (2.0 W)	-

V. EXPERIMENTAL METHODOLOGY

A. Real System Measurement Infrastructure

We perform our experiments on three smartphones—Mi8Pro [49], Galaxy S10e [94], and Moto X Force [81]. Table II summarizes their specifications[6]. Note that we only use the smartphone with DSP rather than that with NPU, since 1) NPUs are only programmable through vendor-provided software development kits (SDKs) which have yet to see public release [51], and 2) DSPs in recent mobile SoCs are optimized for DNN inference so that they can act as NPUs [51], [90].

[5]Note that we use 0.1 for ε by referring to previous RL-based works in this domain [75], [83].

[6]Although mobile processors have lower-performance CPU cores as well, we only present the high-performance ones since they are what DNN inference usually runs on.

TABLE III
DNN INFERENCE WORKLOADS. LAYER COMPOSITIONS ARE OBTAINED FROM THE TENSORFLOW NN IMPLEMENTATIONS.

Workload	DNN	S_{CONV}	S_{FC}	S_{RC}
Image Classification	Inception v1	49	1	0
	Inception v3	94	1	0
	MobileNet v1	14	1	0
	MobileNet v2	35	1	0
	MobileNet v3	23	20	0
	ResNet 50	53	1	0
Object Detection	SSD MobileNet v1	19	1	0
	SSD MobileNet v2	52	1	0
	SSD MobileNet v3	28	20	0
Translation	MobileBERT	0	1	24

For cloud inference execution, we connect the smartphones to a server, equipped with an Intel Xeon CPU E5-2640 with 2.4GHz of 40 cores, an NVIDIA Tesla P100 GPU, and 256 GB of RAM. To control the Wi-Fi signal strength, we adjust the distance between the smartphones and the access point (AP). For inference execution on locally connected edge, we use a tablet, Galaxy Tab S6, equipped with 2.84 GHz of Cortex A76 CPU, an Adreno 640 GPU, and a Hexagon 690 DSP. We connect the smartphones to the tablet through Wi-Fi Direct. To control the signal strength of Wi-Fi Direct, we adjust the distance between the locally connected devices. We measure the system-wide power consumption of the smartphones using an external Monsoon Power Meter [80]; prior works used the similar practice [8], [11], [88].

To execute DNN inference on diverse processors in edge-cloud systems, we build atop TVM [12] and SNPE [90]. TVM compiles NNs from TensorFlow/TF Lite and generates executables for edge/cloud CPUs and GPUs, whereas SNPE compiles NNs and generates executables for mobile DSPs. The executables are deployed onto each device with runtime library implementations of TVM [12] and SNPE [90], enabling edge inference at runtime.

To evaluate the effectiveness of AutoScale[7], we compare it with five baselines in our edge-cloud systems: Edge (CPU FP32), which always runs DNN inference on the CPU of the edge device; Edge (Best), which runs the inference on the most energy efficient processors of the edge device; Cloud, which always runs inference on the cloud; Connected Edge, which always runs inference on another locally connected edge device; and Opt[8], an oracular design that always runs inference on the optimal execution target. We also compare AutoScale with two closely related prior works: MOSAIC [42] and NeuroSurgeon [53].

B. Benchmarks and Execution Scenarios

For our evaluation, we use the 10 neural networks in Table III, which are widely used in real use case scenarios [42], [76], [93], [108]. To explore real use cases, we implement an Android application. For computer vision workloads (i.e., image classification and object detection), we implement two scenarios: non-streaming and streaming. For the non-streaming scenario, the Android application takes an image from the camera and performs inference on it. For this scenario, short response time is important to users. Since users cannot perceive any difference as long as the response time is less than 50 ms [23], [74], [122], we use 50 ms as the QoS target. On the other hand, for the streaming scenario, the Android application takes a real-time video from the camera and performs inference on it. For this scenario, high frames per second (FPS) is important for user satisfaction. Since users cannot perceive any QoS difference as long as the FPS exceeds 30 [22], [122], we use 30 FPS as the QoS target. For MobileBERT in natural language processing, we implement one scenario: the Android application translates a sentence entered by keyboard. We use 100 ms as the QoS target in this case [93].

TABLE IV
DNN INFERENCE EXECUTION ENVIRONMENT.

Environment		Description
Static	S1	No runtime variance
	S2	CPU-intensive co-running app
	S3	Memory-intensive co-running app
	S4	Weak Wi-Fi signal
	S5	Weak Wi-Fi Direct signal
Dynamic	D1	Co-running app: music player
	D2	Co-running app: web browser
	D3	Random Wi-Fi signal
	D4	Varying co-running apps

To validate the effectiveness of AutoScale in real environment with varying runtime variance, we run our experiments in two environments: static and dynamic. For the static environment, we fix the runtime variance (i.e., co-running apps with constant CPU and memory usages and constant Wi-Fi and Wi-Fi Direct signal strengths). For the dynamic environment, we vary the runtime variance. In case of the co-running apps, we use two real-world applications: a web browser and a music player. For the web browser, we encode the series of inputs using an automatic input generator [57] to represent real use cases. In addition, since the signal strength variance is typically modeled by a Gaussian distribution [19], we emulate the random signal strength with a Gaussian distribution by adjusting the bandwidth limit of Wi-Fi AP. We also conduct experiments with varying co-running apps from the music player to the web browser. Table IV summarizes the DNN inference execution environments.

C. AutoScale Design Specification

Actions: We determine the actions of AutoScale with processors available in our edge-cloud system. Since the energy efficiency of mobile CPU/GPU can be further optimized via DVFS, we identify each voltage/frequency (V/F) step as the augmented action; Table II shows the number of available V/F steps. We do not consider DVFS for DSP in our experiments, since DSP does not support DVFS yet. We also identify the quantization available for each mobile processor (INT8 for CPUs and FP16 for GPUs) as the augmented

[7]AutoScale is implemented as part of intelligent services and runs on the mobile CPU. It obtains the NN characteristics using the TVM and SNPE runtime libraries as well as other information (i.e., mobile resource usage and signal strength) through system kernel APIs.

[8]To obtain Opt, we measure the inference latency, accuracy, and energy efficiency for each device over the entire design space of about 200,000 (3,072 states times ∼66 actions augmented with quantization and DVFS). We then define it as the setup that provides the highest energy efficiency while meeting the QoS and accuracy requirements.

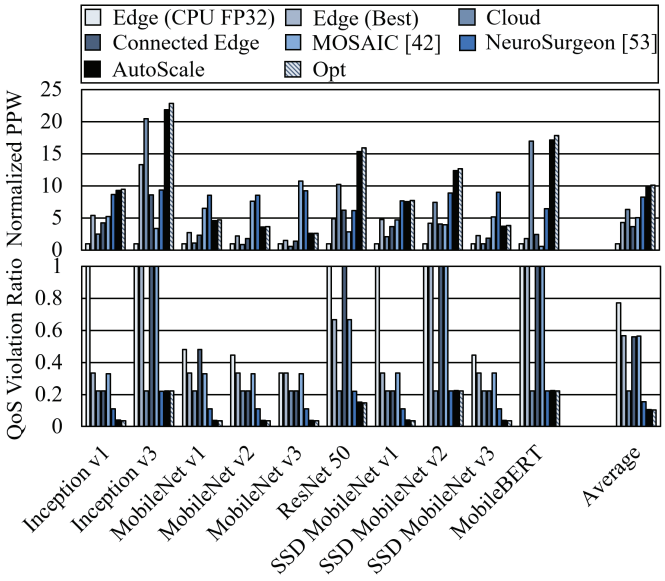


Fig. 9. AutoScale significantly improves energy efficiency compared with the baselines and prior work [42],[53] satisfying QoS constraints.

action. In summary, AutoScale defines the actions for our edge-cloud system as follows: mobile CPU with FP32/INT8, DVFS settings; mobile GPU with FP32/FP16, DVFS settings; mobile DSP; cloud CPU with FP32; cloud GPU with FP32; connected mobile CPU with FP32; connected mobile GPU with FP32; and connected mobile DSP. Note that depending on the configurations of edge-cloud systems, additional actions, such as mobile NPU or cloud TPU, could be further considered.

Hyperparameters: To determine two hyperparameters—the learning rate and discount factor—we evaluate three values of 0.1, 0.5, and 0.9 for each hyperparameter. We observe that a higher learning rate is better, meaning the more the reward is reflected to the Q values, the better AutoScale works. We also observe that a lower discount factor is better. This means that the consecutive states have a weak relationship due to the stochastic nature, so that giving less weight to the rewards in the near future improves the efficiency of AutoScale. Thus, in our evaluation, we use 0.9 for the learning rate and 0.1 for the discount factor.

Training and Testing: To cover the design space of AutoScale with sufficient training samples, we repeatedly run inference 100 times for each NN in each runtime variance-related state (i.e., S_{CO_CPU}, S_{CO_MEM}, S_{RSSI_W}, and S_{RSSI_P} in Table I). Section VI-C provides our analysis on the training overhead. For testing, we use the leave-one-out cross-validation method across the NNs in Table III [115]; for testing each NN, we used a Q-table trained with the rest of NNs.

VI. Evaluation Results and Analysis

A. Performance and Energy Efficiency

Fig. 9 shows the average energy efficiency (PPW) normalized to Edge (CPU FP32) as well as the QoS violation ratio of DNN inference on three mobile devices in static environments. Overall, AutoScale improves the average energy efficiency of the inference by 9.8x, 2.3x, 1.6x, and 2.7x compared to

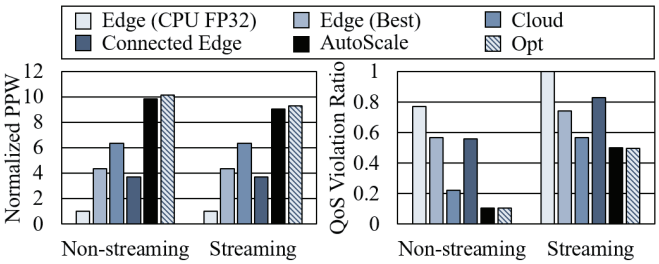


Fig. 10. Even when the inference intensity rises (from non-streaming to streaming), AutoScale still substantially increases the energy efficiency and exhibits much lower QoS violation ratio compared to the baselines.

Edge (CPU FP32), Edge (Best), Cloud, and Connected Edge, respectively. Across the diverse neural networks, AutoScale predicts the optimal execution target to maximize the DNN inference energy efficiency, satisfying the QoS constraint as much as possible. AutoScale achieves almost the same energy efficiency improvement as Opt; the difference is only 3.2%.

In addition, AutoScale exhibits a considerably lower QoS violation ratio compared to the baselines. In fact, it achieves nearly the same ratio as Opt; the difference is only 1.9%. For light NNs, AutoScale does not violate the QoS constraint except when CPU- and memory-intensive applications are co-running or the wireless network signal strength is weak. For heavy NNs, AutoScale mostly relies on cloud execution, so that QoS violation occurs when the Wi-Fi signal strength is weak.

Compared to MOSAIC [42] and NeuroSurgeon [53], AutoScale also improves the average energy efficiency by 1.9x and 1.2x, respectively. Those two techniques offload computations at the granularity of NN layers, whereas AutoScale does so at a coarser, model granularity. Furthermore, both MOSAIC [42] and NeuroSurgeon [53] are based on simple regression models. As Section III-C shows, such approaches often fail to capture stochastic runtime variance, such as on-device interference and signal strength variation. AutoScale accurately predicts the optimal execution target by adapting to the runtime variance using RL. Hence, it shows higher energy efficieny compared to the prior work, satisfying the QoS constraint.

When the inference intensity rises (from non-streaming to streaming scenario), the energy efficiency and QoS violation ratio of AutoScale is degraded, as shown in Fig. 10. Nevertheless, since AutoScale accurately selects the optimal execution target regardless of the inference intensity, it achieves almost the same energy efficiency and QoS violation ratio as Opt.

B. Adaptability and Accuracy Analysis

Adaptability to Stochastic Variance: Fig. 11 shows the average energy efficiency normalized to Edge (CPU FP32) and the QoS violation ratio of DNN inference in the presence of stochastic variance. The x-axis represents the inference environments (Table IV). Since AutoScale accurately predicts the optimal execution scaling even with stochastic variance, it improves the average inference energy efficiency by 10.7x, 2.2x, 1.4x, and 3.2x compared with Edge (CPU FP32), Edge (Best), Cloud, and Connected Edge, respectively, while showing a similar QoS violation ratio as Opt.

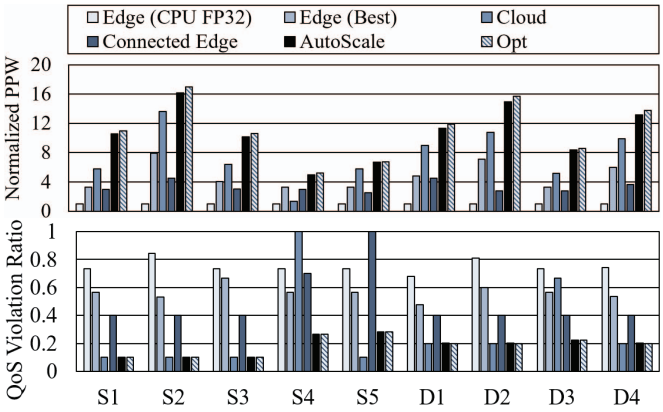


Fig. 11. Since AutoScale accurately predicts the optimal target in the presence of stochastic variance, it largely improves the energy efficiency of DNN inference in realistic environments while satisfying the QoS target.

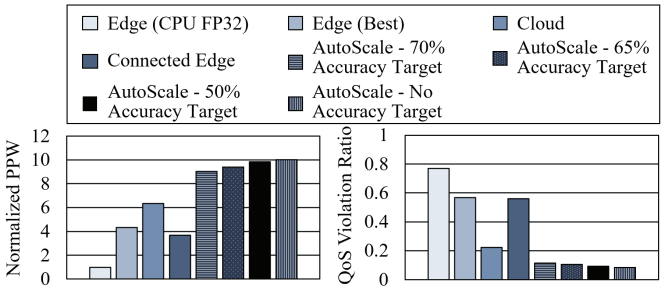


Fig. 12. When AutoScale uses a higher accuracy target, its energy efficiency and QoS violation ratio are slightly degraded. Nevertheless, it still significantly improves the energy efficiency compared to the baselines.

Adaptability to Inference Quality Targets: Fig. 12 shows the average energy efficiency and QoS violation ratio for different inference accuracy targets under AutoScale. When AutoScale uses a high inference accuracy target (i.e., 70% or 65%), it avoids choosing the on-device processors with low precision. On the other hand, when it relaxes the inference accuracy target to 50% or none, it chooses on-device execution with low precision where some NN inference results in lower accuracy. Thus, when AutoScale uses lower accuracy targets, its energy efficiency and QoS violation ratio are improved. The improvement does not vary much beyond the 50% accuracy threshold because the most energy efficient target usually offers higher inference accuracy than 50% in our setup.

Prediction Accuracy: To analyze the prediction accuracy of AutoScale, we compare its execution scaling decision to the optimal one. Fig. 13 shows how AutoScale and Opt make the decision on three mobile devices.

AutoScale accurately selects the optimal execution scaling decision for all devices, achieving 97.9% prediction accuracy on average. It mis-predicts the optimal target only when the energy difference between the optimal target and the (mis-predicted) sub-optimal target is less than 1%. This is owing to the small R_{energy} error. Although AutoScale makes a sub-optimal choice in a few cases, it does not much degrade the overall system energy efficiency and QoS violation ratio compared to Opt. This is due to the small energy difference between the optimal and sub-optimal ones.

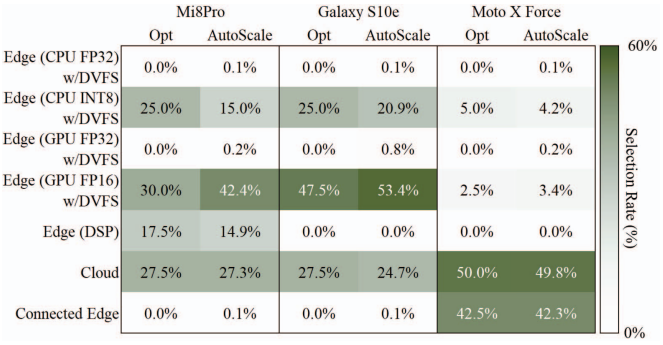


Fig. 13. AutoScale accurately selects the optimal execution target.

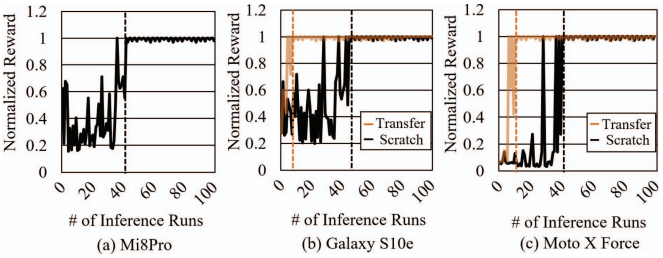


Fig. 14. The reward is usually converged in 40–50 runs. A learning transfer can accelerate the convergence.

Even in the presence of runtime variance, AutoScale accurately makes the optimal execution scaling decision. For example, when the signal strength gets weaker (S4 in Table IV), AutoScale selects on-device inference (69.1%) or connected edge (30.7%) rather than cloud (0.2%), achieving 97.8% prediction accuracy. On the other hand, when the web browser is co-running (D2 in Table IV), AutoScale selects cloud (46.1%) or connected edge (35.3%) rather than on-device inference (18.6%), achieving 97.3% accuracy.

C. Overhead Analysis

Training Overhead: Fig. 14 shows that when training a model from scratch, the reward converges after about 40–50 inference runs on average. Before convergence, AutoScale exhibits 18.9% lower average energy efficiency than Opt. Nevertheless, it still achieves 66.1% energy saving against Edge (CPU FP32). The training overhead can be alleviated with learning transfer. As shown in Figure 14, when the model trained on the Mi8Pro is used for the Galaxy S10e and Moto X Force, the training converges more rapidly, reducing the average training time overhead by 21.2%. This result implies AutoScale can capture the common characteristics across the variety of edge inference workloads, system performance and power profiles, and environmental uncertainties. Note that in dynamic environments (i.e., D1–D4 in Table IV), the reward converges 9.1% slower than in the stable environments (i.e., S1–S4 in Table IV). Nonetheless, this long convergence in dynamic environments can also be alleviated with learning transfer; when the model trained on the Mi8Pro is used for the Galaxy S10e and Moto X Force, training in dynamic environments also converges rapidly, so that the convergence time difference between dynamic and stable environments decreases to 0.5%.

Runtime Overhead: To demonstrate the viability of mobile inference deployment, we evaluate the AutoScale runtime overhead. The performance overhead of RL algorithm in AutoScale is, on average, 25.4 μs for training, excluding the time for inference execution. It corresponds to 1.2% of the lowest inference latency. In addition, when using the trained Q-table, the overhead can be reduced to 7.3 μs with only 0.3% overhead. This result means it takes 18.1 μs to measure the inference results, calculate the reward, and update the Q-table. The energy overhead is only 1.0% and 0.2% of the total system energy consumption, when training the Q-table and exploiting the trained Q-table, respectively. The memory requirement of AutoScale is 0.4 MB, translating to only 0.01% of the 3 GB DRAM capacity of a typical mid-end mobile device [81].

VII. Related Work

With the emergence of DNN-based intelligent services, energy optimization of mobile DNN inference has been widely studied. Due to the compute- and memory-intensive nature, many of the early works executed DNN inference in the cloud [13], [28], [56], [73]. As mobile systems become powerful [31], [41], [50], [107], there has been increasing push to execute DNN inference at the edge [10], [25], [42], [53], [55], [65], [106], [107], [111], [121]. As an intermediate stage, many techniques tried to partition DNN inference execution between the cloud and the local mobile device [24], [25], [40], [53], [69], using performance/energy prediction models. These techniques, however, do not consider fully executing inference at the edge. According to our analysis, there exist various cases where edge inference outperforms cloud inference by removing data transmission overhead. More importantly, previous techniques also do not consider stochastic variance, such as on-device interference and signal strength variation, which largely affects inference efficiency.

To execute DNN inference entirely at the edge, many optimizations have been proposed, including model architecture search [95], [104], [110], [124], quantization [15], [30], [52], [62], [65], [111], [119], weight compression [18], [43], [68], [72], and graph pruning [112], [118]. In addition, deep learning compiler and programming stacks have been improved to ease the adoption of co-processors. On top of these works, many researchers tried to optimize the performance and/or energy efficiency of edge inference execution by exploiting co-processors or near-sensor processing along with CPUs [10], [42], [55], [63], [65], [106], [107], [121]. However, most of the above techniques are based on existing prediction approaches, which are prone to being affected by stochastic variance. Moreover, they also do not consider executing inference on connected systems, such as the cloud server or a locally connected mobile device.

Considering the uncertainties of the mobile environment, various energy management techniques have been proposed [33], [34], [98]. To maximize smartphone energy efficiency subject to user satisfaction demands under the memory interference, DORA takes a regression-based predictive approach to control the frequency settings of mobile CPUs at runtime [98]. Gaudette et al. proposed to use arbitrary polynomial chaos expansions to consider the impact of various uncertainties on mobile user experience [34]. Various offloading techniques that execute all or some computations on remote servers while considering the variability of the mobile environment have been also proposed [1], [2], [16], [48], [56], [61], [87], [91], [113], [117]. Some of these techniques employed reinforcement learning (RL) to handle network variability [1], [48], [113], [117], dependency between mobile workloads [87], or server resource utilization [91]. The above techniques, however, are not directly applicable to energy efficiency optimization of edge inference, since they do not consider the characteristics of NN inference workloads. Other works have employed RL to handle runtime variance and/or a large design space for web browsers, latency-sensitive cloud services, data center job scheduling, and NN code generations for heterogeneous processors [12], [14], [26], [75], [83].

To the best of our knowledge, this is the first work that demonstrates the potential of DNN inference at the edge by automatically leveraging co-processors as well as other computing resources nearby and in the cloud. We examine a collection of machine learning-based predictive approaches, and tailor-design an automatic execution scaling engine with lightweight, customized reinforcement learning. AutoScale achieves near-optimal energy efficiency for edge inference while adapting to stochastic variance, particularly important for user quality of experience in the mobile domain.

VIII. Conclusion

Given the growing ubiquity of intelligent services, including virtual assistant, face/image recognition, and language translation, deep learning inference is increasingly run at the edge. To enable energy efficient inference at the edge, we propose an adaptive and light-weight deep learning execution scaling engine—AutoScale. The in-depth characterization of DNN inference on mobile and edge-cloud systems demonstrates that the optimal scaling decision depends on various features: NN characteristics, QoS and accuracy targets, underlying system profiles, and stochastic runtime variance. AutoScale continuously learns and selects the optimal execution scaling decision by taking into account the features and dynamically adapting to the stochastic runtime variance. We design and construct representative edge inference use cases and mobile-cloud execution environment using off-the-shelf systems. On average, AutoScale improves DNN inference energy efficiency by 9.8x and 1.6x, as compared to the baseline settings of mobile CPU and cloud offloading, satisfying both the QoS and accuracy constraints. We demonstrate that AutoScale is a viable solution and will pave the path forward by enabling future work on energy efficiency improvement for DNN edge inference in a variety of realistic execution environments.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under SHF-1652132, CCF-1618039, and CCF-1525462 for Young Geun Kim and Carole-Jean Wu at ASU.

REFERENCES

[1] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[2] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy cost models of smartphones for task offloading to the cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, 2015.

[3] Amazon, "Alexa." [Online]. Available: https://developer.amazon.com/en-US/alexa

[4] Android, "Android neural networks api." [Online]. Available: https://developer.android.com/ndk/guides/neuralnetworks

[5] Apple, "Siri." [Online]. Available: https://www.apple.com/siri

[6] J. I. Benedetto, L. A. Gonzalez, P. Sanabria, A. Neyem, and J. Navon, "Towards a practical framework for code offloading in the internet of things," *Future Generation Computer Systems*, vol. 92, 2019.

[7] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Bechmark analysis of representative deep nerual network architectures," *IEEE Access*, vol. 6, no. 1, pp. 64 270–64 277, 2018.

[8] W. L. Bircher and L. K. John, "Complete system power estimation: A trickle-down approach based on performance events," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2007, pp. 171–180.

[9] Caffe2, "A new lightweight, modular, and scalable deep learning framework." [Online]. Available: https://caffe2.ai

[10] E. Cai, D.-C. Juan, D. Stamoulis, and D. Maculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Proceedings of the Asian Conference on Machine Learning (ACML)*, 2017.

[11] A. Caroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the USENIX Annual Technical Conference*, 2010.

[12] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[13] Y. Chen, J. Hen, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 73–82.

[14] Y. Choi, S. Park, and H. Cha, "Optimizing energy efficiency of browsers in energy-aware scheduling-enabled mobile devices," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv:1602.02830v3*, 2016.

[16] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphone last longer with code offload," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[17] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009.

[18] C. Ding, Y. Wang, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan, "Circnn: Accelerating and compressing deep neural networks using block-circulantweight matrices," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017, pp. 395–408.

[19] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013, pp. 29–40.

[20] B. Donyanavard, A. Sadighi, T. Muck, F. Maurer, A. M. Rahmani, A. Herkersdorf, and N. Dutt, "Sosa: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019, pp. 685–698.

[21] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 1997.

[22] B. Egilmez, M. Schuchhardt, G. Memik, R. Ayoub, N. Soundararajan, and M. Kishinevsky, "User-aware frame rate management in android smartphones," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 1–17, 2017.

[23] Y. Endo, Z. Wang, J. B. Chen, and M. I. Seltzer, "Using latency to evaluate interactive system performance," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1996.

[24] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, 2020.

[25] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of Great Lakes Symposium on VLSI (GVLSI)*, 2018, pp. 111–116.

[26] R. Evans and J. Gao, "Deepmind ai reduces google data centre cooling bill by 40%." [Online]. Available: https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40

[27] E. Even-Dar, S. Mannor, and Y. Mansour, "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems," *Journal of Machine Learning Research*, vol. 7, pp. 1079–1105, 2006.

[28] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "Qos-aware scheduling of heterogeneous servers for inference in deep neural networks," in *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2017, pp. 2067–2070.

[29] Fitbit, "Fitbit flex 2." [Online]. Available: https://www.fitbit.com/in/flex2

[30] J. Fromm, M. Cowan, M. Philipose, L. Ceze, and S. Patel, "Riptide: Fast end-to-end binarized neural networks," in *Proceedings of Machine Learning and Systems (MLSys)*, 2020.

[31] C. Gao, A. Gutierrez, M. Rajan, R. Dreslinski, T. Mudge, and C.-J. Wu, "A study of mobile device utilization," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

[32] J. R. Gardner, M. J. Kusner, Z. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

[33] B. Gaudette, C.-J. Wu, and S. Vrudhula, "Improving smartphone user experience by balancing performance and energy with probabilistic qos guarantee," 2016.

[34] B. Gaudette, C.-J. Wu, and S. Vrudhula, "Optimizing user satisfaction of mobile workloads subject to various sources of uncertainties," *IEEE Transactions on Mobile Computing*, vol. 18, no. 12, pp. 2941–2953, 2019.

[35] Google, "Google cardboard." [Online]. Available: https://arvr.google.com/cardboard/

[36] Google, "Google cloud vision." [Online]. Available: https://cloud.google.com/vision

[37] Google, "Google daydream." [Online]. Available: https://arvr.google.com/daydream/

[38] Google, "Google translate." [Online]. Available: https://translate.google.com

[39] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE Access*, vol. 8, pp. 13 937–13 948, 2020.

[40] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *Proceedings of International Conference on Cloud Engineering (IC2E)*, 2018, pp. 184–190.

[41] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 64–76.

[42] M. Han, J. Hyun, S. Park, J. Park, and W. Baek, "Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2019, pp. 165–177.

[43] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[44] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *Proceedings of the IEEE International Symposium on Computer Architecture (ISCA)*, 2015.

[45] 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 (CVPR)*, 2016.

[46] C. Healthcare, "The intelligent healthcare platform." [Online]. Available: https://www.changehealthcare.com/about/innovation/intelligent-healthcare-platform

[47] A. Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019.

[48] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, 2020.

[49] Huawei, "Kirin 980, the world's first 7nm process mobile ai chipset." [Online]. Available: https://consumer.huawei.com/en/campaign/kirin980/

[50] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2017, pp. 82–95.

[51] A. Ignatov, R. Timofte, W. Chou, K. Wang, T. Hartley, and L. V. Gool, "AI benchmark: Running deep neural networks on android smartphones," *arXiv:1810.01109*, 2018.

[52] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[53] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 615–629.

[54] A. Karpathy, G. Toderici, S. Shetty, T. Leung, A. Sukthankar, and L. Fei-fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1725–1732.

[55] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "ulayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2019.

[56] Y. G. Kim and S. W. Chung, "Signal strength-aware adaptive offloading for energy efficient mobile devices," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2017, pp. 1–6.

[57] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1878–1889, 2017.

[58] Y. G. Kim, M. Kim, J. M. Kim, M. Sung, and S. W. Chung, "A novel gpu power model for accurate smartphone power breakdown," *ETRI Journal*, vol. 37, no. 1, pp. 157–164, 2015.

[59] Y. G. Kim, M. Kim, J. Kong, and S. W. Chung, "An adaptive thermal management framework for heterogeneous multi-core processors," *IEEE Transactions on Computers*, vol. 69, pp. 894–906, 2020.

[60] Y. G. Kim, J. Kong, and S. W. Chung, "A survey on recent os-level energy management techniques for mobile processing units," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2388–2401, 2018.

[61] Y. G. Kim, Y. S. Lee, and S. W. Chung, "Signal strength-aware adaptive offloading with local image preprocessing for energy efficient mobile devices," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 99–101, 2020.

[62] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay, "Adaptive weight compression for memory-efficient neural networks," in *Proceedings of Design, Automation, and Test in Europe Conference (DATE)*, 2017.

[63] V. Kodukula, S. Katrawala, B. Jones, C.-J. Wu, and R. LiKamWa, "Stagioni: Temperature management to enable near-sensor processing for energy-efficient high-fidelity imaging," *arXiv:2001.01580*, 2019.

[64] D. E. Koulouriotis and A. Xanthopoluos, "Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems," *Applied Mathematics and Computation*, vol. 196, 2008.

[65] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, 2016, pp. 98–107.

[66] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2009, pp. 609–616.

[67] S.-Y. Lee and C.-J. Wu, "Performance characterization, prediction, and optimization for heterogeneous systems with multi-level memory interference," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2017, pp. 43–53.

[68] D. Li, X. Wang, and D. Kong, "Deeprebirth: Accelerating deep neural network execution on mobile devices," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[69] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, and X. Feng, "Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge," in *Proceedings of International Conference on Artificial Neural Networks (ICANN)*, 2018.

[70] X. Lin, Y. Wang, and M. Pedram, "A reinforcement learning-based power management framework for green computing data centers," in *Proceedings of the International Conference on Cloud Engineering (IC2E)*, 2016, pp. 135–138.

[71] J. Liu, W. C. Chang, Y. Wu, and Y. Yang, "Deep learning for multi-label text classification," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2017, pp. 115–124.

[72] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2018.

[73] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing cnn model inference on cpus," in *Proceedings of the USENIX Annual Technical Conference*, 2019, pp. 1025–1039.

[74] D. Lo, T. Song, and G. E. Suh, "Prediction-gudided performance-energy trade-off for interactive applications," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015, pp. 508–520.

[75] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "An energy-aware online learning framework for resource management in heterogeneous platforms," *ACM Transactions on Design Automation and Electronic Systems*, 2020.

[76] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G. Wei, and C.-J. Wu, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, 2020.

[77] Microsoft, "Azure artificial intelligence." [Online]. Available: https://azure.microsoft.com/en-us/free/ai/

[78] Microsoft, "Hololens2." [Online]. Available: https://www.microsoft.com/en-us/hololens

[79] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattile, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wiersta, S. Legg, and D. Hassabis, "Human-level control thourgh deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[80] Monsoon, "High voltage power monitor." [Online]. Available: https://www.msoon.com/high-voltage-power-monitor

[81] Motorola, "Moto x force - technical specs." [Online]. Available: https://support.motorola.com/uk/en/solution/MS112171

[82] MXNet, "A flexible and efficient library for deep learning." [Online]. Available: https://mxnet.apache.org/

[83] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid task manager for latency-critical cloud workloads," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 409–420.

[84] Oculus, "Turn the world into your arcade." [Online]. Available: https://www.oculus.com/?locale=en_US

[85] OMRON, "Healthguide - blood pressure monitoring anytime, anywhere." [Online]. Available: https://omronhealthcare.com/products/heartguide-wearable-blood-pressure-monitor-bp8000m/

[86] S. Pagani, S. Manoj, A. Jantsch, and J. Henkel, "Machine learning for power, energy, and thermal management on multicore processors: A survey," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 101–116, 2020.

[87] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng, "Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 134 742–134 753.

[88] D. Pandiyan and C.-J. Wu, "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 171–180.

[89] PyTorch, "From research to production: An open source machine learning framework that accelerates the path from research prototyping to production deployment." [Online]. Available: https://pytorch.org/

[90] Qualcomm, "Snapdragon neural processing engine sdk." [Online]. Available: https://developer.qualcomm.com/docs/snpe/overview.html

[91] L. Quan, Z. Wang, and F. Ren, "A novel two-layered reinforcement learning for task offloading with tradeoff between physical machine utilization rate and delay," *Future Internet*, vol. 10, pp. 1–17.

[92] B. Reagen, J. M. Hernandez-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G. Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2017.

[93] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Mcikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeria, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *arXiv:1911.02549*, 2019.

[94] Samsung, "Samsung galaxy s10e, s10, & s10+." [Online]. Available: https://www.samsung.com/global/galaxy/galaxy-s10

[95] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[96] G. A. F. Seber and A. J. Lee, *Linear Regression Analysis*, 2nd ed. John Wiley & Sons, 2012.

[97] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 1–32, 2013.

[98] D. Shingari, A. Arunkumar, B. Gaudette, S. Vrudhula, and C.-J. Wu, "Dora: Optimizing smartphone energy efficiency and web browser performance under interference," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 64–75.

[99] Statista, "Forecast number of mobile users worldwide from 2019 to 2023." [Online]. Available: https://statista.com/statistics/218984/number-of-global-mobile-users-since-2010

[100] Statista, "Number of connected wearable devices worldwide by region from 2015 to 2022." [Online]. Available: https://www.statista.com/statistics/490231/wearable-devices-worldwide-by-region/

[101] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: A compact task-agnostic bert for resource-limited devices," *arXiv:2004.02984*, 2020.

[102] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, pp. 293–300, 1999.

[103] C. Szeged, 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 (CVPR)*, 2015, pp. 1–9.

[104] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946*, 2019.

[105] Tensorflow, "An end-to-end open source machine learning platform." [Online]. Available: https://www.tensorflow.org/

[106] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput cnn inference on embedded arm big.little multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[107] S. Wang, A. Pathania, and T. Mitra, "Neural network inference on mobile socs," *IEEE Design & Test*, 2020.

[108] Y. E. Wang, C.-J. Wu, X. Wang, K. Hazelwood, and D. Brooks, "Exploiting parallelism opportunities with deep learning frameworks," *arXiv:1908.04705*, 2019.

[109] WITHINGS, "The world's first analog watch with a built-in electrocardiogram to detect atrial fibrillation." [Online]. Available: https://www.withings.com/us/en/move-ecg?utm_source=CJ&utm_medium=Affiliate&utm_campaign=affiliation-Skimlinks&utm_content=7099101-Home+Page+US-13184200&CJEVENT=3669c89e7f8511ea83c700830a1c0e13

[110] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *arXiv:1812.03443*, 2018.

[111] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.

[112] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Proceedings of the IEEE International Symposium on Computer Architecture (ISCA)*, 2017, pp. 548–560.

[113] B. Zhang, G. Zhang, W. Sun, and K. Yang, "Task offloading with power control for mobile edge computing using reinforcement learning-based markov decision process," *Mobile Information Systems*, 2020.

[114] B. Zhang and S. N. Srihari, "Fast k-nearest neighbor classification using cluster-based trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 4, pp. 525–528, 2004.

[115] K. Zhang, A. Guliani, S. O.-. Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine learning-based temperature prediction for runtime thermal management across system components," vol. 29, no. 2, pp. 405–419, 2018.

[116] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2010, pp. 105–114.

[117] T. Zhang, Y. H. Chiang, C. Borcea, and Y. Ji, "Learning-based offloading of tasks with diverse delay sensitivities for mobile edge computing," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2019.

[118] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.

[119] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[120] R. Zhao and V. Tresp, "Efficient dialog policy learning via positive memory retention," in *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, 2018.

[121] G. Zhong, A. Dubey, C. Tan, and T. Mitra, "Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 2, pp. 1–23, 2019.

[122] Y. Zhu, M. Halpern, and V. J. Reddi, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 137–149.

[123] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 13–24.

[124] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.