MOC: Multi-Objective Mobile CPU-GPU Co-optimization for Power-efficient DNN Inference

Yushu Wu*[1], Yifan Gong*[1], Zheng Zhan[1], Geng Yuan[2], Yanyu Li[1], Qi Wang[1], Chao Wu†[1], Yanzhi Wang[1]
Northeastern University[1], *University of Georgia*[2]
{wu.yushu,gong.yifa,zhan.zhe,li.yanyu,qi.wa,cha.wu,yanz.wang}@northeastern.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323882

***Abstract*—With the emergence of DNN applications on mobile devices, plenty of attention has been attracted to their optimization. However, the impact of DNN inference tasks on device power consumption is still a lack of comprehensive study. In this work, we propose MOC, a Multi-Objective deep reinforcement learning-assisted DNN inference stage-adaptive CPU-GPU Co-optimization approach. We find through experiments that CPU-GPU parameters, including CPU core, CPU, and GPU frequency, could significantly impact the speed and power consumption of DNN inference. We empirically analyze various stages of DNN inference, including pre/post-processing and feed-forward calculating stages. Based on the analysis, a DNN demand-resource matching model is proposed to classify the DNNs into various categories. Next, a multi-objective deep reinforcement learning (MODRL)-assisted framework is proposed, which considers both the DNN type and hardware environment, to make decisions on DNN inference stage-adaptive CPU/GPU parameter tuning. Finally, a rule-based action refinement technique is introduced to tailor the search space of MOC. Extensive experiments show that, compared with existing works, MOC could substantially reduce the power consumption of DNN inference tasks by up to 74.4%, meanwhile delivering an excellent speed on mobile devices.**

I. Introduction

Given the popularity and widespread adoption of mobile devices, the demands for deploying deep neural networks (DNNs) on these resource-limited platforms has been steadily increasing. A key challenge during the DNN deployment on mobile devices arises from the energy efficiency aspect [1] as they typically operate on limited battery power within a charging cycle. Plenty of research works have delved into exploring various compression techniques [2]–[14] such as pruning and quantization to improve the energy efficiency by reducing the computational load. Yet few works investigate the possibility of reducing the power consumption of DNN inference via appropriate CPU-GPU configuration for DNN inference on the heterogeneous CPU-GPU system.

While offering promising potentials for power efficiency, optimizing CPU-GPU configurations for DNN inference poses a considerable challenge. The optimization entails exploring an extensive search space encompassing different combinations of diverse DNN models and various CPU-GPU configurations. From the perspective of DNN models, their diversity is manifested in both patterns and resource demands. Computation-intensive models, e.g., ResNet [15], are featured with small network parameters which reside long in the on-chip cache for repeated calculation, thus demanding high computational resources. In contrast, memory-intensive DNN models, e.g., WDSR-based super resolution [16], represent high demands on memory resources, as the network parameters could go beyond the capacity of on-chip cache and generate excessive data movement between DRAM and GPU. As different components in the device contribute distinctively in terms of speed and power consumption [17]–[19], it is essential to perform adaptive resource allocation via CPU-GPU parameters tuning. Moreover, the DNN inference process contains several stages, including pre-processing, feed-forward calculation, and post-processing stages. Each stage might have different data operations, hence contributing distinctively to the DNN inference process. As a result, it is challenging to cover various DNN patterns and resource demands for multiple DNN inference stages in CPU-GPU configuration.

From the CPU-GPU configuration perspective, modern mobile is typically equipped with a heterogeneous CPU-GPU system together with different execution frequency levels. More specifically, most current mobile devices are facilitated with ARM big.LITTLE heterogeneous CPU processing architectures [20], together with GPU processors to provide rich computation capacity. Different CPU core clusters and GPU are equipped with multiple frequency levels. In DNN inference, the speed and power consumption could be influenced by the selection of the CPU core cluster, CPU frequency, and GPU frequency. Prior wisdom addresses this issue by enhancing the dynamic voltage and frequency scaling (DVFS) technique but ignores CPU core selection [21]. In our experiments, inactivating parts of CPU cores could reduce the power consumption of ResNet50 by 63.6% while delivering a close speed. The configurable CPU-GPU parameters further complicate the CPU-GPU configuration task.

Due to the huge search space, heuristic tuning methods fail to find the best-suited CPU-GPU configuration efficiently. Some works propose to allocate the hardware resources with reinforcement learning (RL) [22], [23], which mainly focus on the feed-forward calculation stage in DNN inference and perform a coarse-grained configuration with less efficiency. It is necessary to propose a novel approach to find the best-suited CPU-GPU configuration accurately and automatically.

In this work, we propose MOC, a Multi-Objective deep RL-assisted DNN inference stage-adaptive CPU-GPU Co-optimization framework. First, we comprehensively study the DNN inference process and propose a DNN demand-resource

*Equal contribution
†Corresponding author

matching model to classify the DNN models into several categories. Next, a multi-objective deep reinforcement learning (MODRL) model is established, considering the DNN types and the real-time hardware status in CPU-GPU configuration. Finally, a rule-based action refinement technique is proposed, which tailors the search space of MOC by refining the unnecessary CPU-GPU configurations. Combining these techniques, MOC could reach the Pareto-optimal speed and power for given DNN models on mobile devices. Experimental results show that, compared with existing works, MOC could reduce the power consumption of DNN inference by up to 74.4.% with only 1.2% latency degradation.

In summary, this work makes the following contributions.

- We investigated the various stages of DNN inference and proposed a DNN demand-resource matching model to classify DNN models into several categories.
- We proposed MOC, a multi-objective deep reinforcement learning-assisted model to configure the CPU-GPU system adaptive to various DNN types and stages.
- We introduced a rule-based action refinement technique to tailor the search space of MOC, hence guiding MOC to converge to the best-suited CPU-GPU configurations faster and more effectively.

II. Background and Related Work

DNN Inference: Understanding DNN inference process is critical in CPU-GPU parameter configuration. There are three main sequential stages in a DNN inference process, including *pre-processing*, *feed-forward calculation*, and *post-processing* stages. Figure 1 shows the data flow of DNN inference.

- **Pre-processing** is mainly executed on CPU. Initially, the INPUT data is loaded to DRAM. CPU needs to transform the data for GPU calculation. This could incur DRAM-CPU data movement and CPU operations. Finally, the processed data are fed to GPU either in data copy mode via shared memory, or in mapping mode.
- **Feed-forward calculation** is mainly executed on GPU, except for some CPU instructions, such as kernel initialization. Initially, WEIGHT is stored in shared memory, then copied to GPU cache layer-wisely and flushed after computation. Meanwhile, FEATURE MAP is generated as the output feature of the former layer, and copied from GPU cache to shared memory for further use. In calculation, it is copied to GPU cache and fed to the current layer. Thereby, FEATURE MAP is circulated frequently between shared memory and GPU cache.
- **Post-processing** transforms the OUTPUT data to user-perceivable format. OUTPUT is the last feature map of a DNN model. It is generated at GPU cache, transmitted to shared memory, and copied in DRAM, finally transmitted to CPU for post-processing. The post-processing stage could be deemed as a reverse operation of the *pre-processing* stage. Thus, a similar resource demand should be shared by both stages.

Different stages have distinguished resource demands. The pre/post-process stages are sensitive to the bandwidth of CPU caches and DRAM. To boost this stage, the system should improve the CPU DVFS level, or enlarge the CPU cache by activating bigger CPU core cluster. In contrast, the feed-forward calculation stage holds an apparent reliance on the GPU capacity for fast calculation.

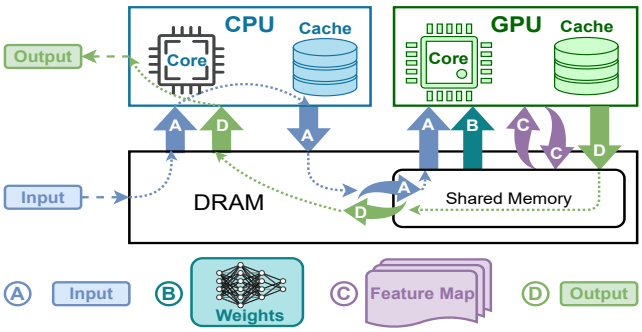


Fig. 1: Data Flow during DNN Inference.

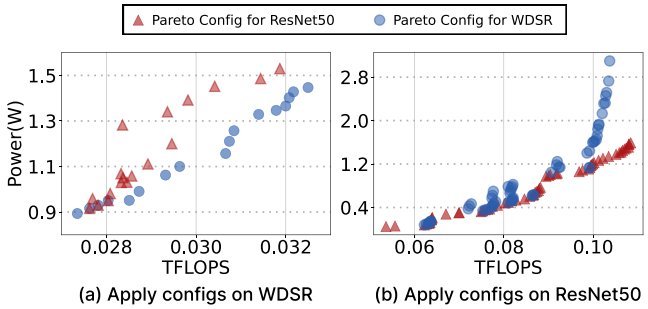
Heterogeneous CPU-GPU System: Mobile devices are mostly equipped with ARM big.LITTLE heterogeneous CPU architectures [20]. For example, the Snapdragon CPU platform in Oneplus 8T [24] is equipped with 1 big core, 3 medium cores, and 4 little cores. The big core is power-hungry with high performance, while the little cores are the opposite. Moreover, multiple voltage and frequency levels are provided for both CPU and GPU to be selected. Hence, it is prohibitively expensive to manually search for the best-suited configuration in such a myriad searching space.

Prior arts leverage the DVFS technique to trade off the power and performance of the device. A higher voltage provides better performance for intensive workloads, e.g., mobile games, at the cost of increased power consumption [25]. Some works study the impact of CPU [26]–[29] or GPU [30]–[32] DVFS on DNNs. Other works [22], [23] propose to allocate the hardware resources for DNN inference tasks through the RL model. However, they are less efficient for three reasons. First, they perform a coarse-grained allocation, focusing only on the feed-forward calculating stage. Instead, MOC configures the CPU-GPU system for each stage in DNN inference. Second, they simply feed the parameters of DNN network and hardware to the RL model, leading to a long training time. In contrast, MOC classifies DNN models into several categories and takes the DNN type as the input. Moreover, we empirically set several rules for action refinement in MOC. Hence, the training process of MOC could be significantly shortened and converged to stable status. Third, they adopt the single-objective RL model, with a constrained power or speed in their rewards. This could lead to the convergence of local optimality. MOC strives to reach the global-optimal Pareto-frontiers by adopting a multi-objective RL technique.

In summary, current works targeting DNN inference-oriented hardware resource allocation are less efficient. This poses the necessity of a novel framework in DNN inference-adaptive CPU-GPU co-optimization.

stage	Cluster	#core	f_{CPU}	f_{GPU}	T(ms)	ΔP(W)
P	medium	3	1.8GHz	N/A	7.72	0.78
P	little	4	1.8GHz	N/A	12.97	0.75
F	medium	3	1.8GHz	587MHz	28.47	2.19
F	little	4	1.8GHz	587MHz	29.19	1.97

TABLE I: Impact of CPU-GPU configurations on DNN inference stages (WDSR). **P** denotes the sum of pre- and post-processing stage, **F** denotes feed-forward calculating stage.



(a) Apply configs on WDSR (b) Apply configs on ResNet50

Fig. 2: a) Applying both configurations including ResNet50 Pareto-optimal configurations on WDSR. b) Applying both configurations including WDSR Pareto-optimal configurations on ResNet50. Red triangles indicate those configurations are ResNet50's Pareto-optimal configuration. Blue circles are the Pareto-optimality for WDSR.

III. Motivation

A. Impact of CPU-GPU Configs on DNN Inference Stages

In DNN inference, the three stages are performed sequentially, and the inference time could be obtained by summing up the time of each stage. It is important to note that different stages of DNN inference favor different configurations, as they are mainly executed on distinct processors. Tab. I shows the impact of CPU-GPU configurations on DNN inference stages. As pre- and post-processing stages reveal similar resource demands, we merge them together. For **P** stage, changing the CPU configuration will incur latency degradation by 68.0% while only gaining 3.9% power reduction. However, for **F** stage, similar configuration tuning only incurs 2.5% latency degradation but obtains 11.2% power reduction. These results indicate that different configurations for different stages could further optimize the power consumption of DNN inference.

B. Demand of DNN-adaptive System Configurations

DNN inference tasks are diverse in hardware resource demands. To demonstrate the diversity, we select three DNN models with different patterns, including ResNet50 [15], WDSR [16], and EfficientNet-b0 [33] as the representatives. Tab. II describes the details of the selected DNNs. With the diverse models in hand, we wonder if there is a once-for-all best-suited configuration to achieve power-efficient DNN inference. To verify this, we manually evaluate the speed and power of one DNN model i to get the best-suited configuration c_i^*. Then, this configuration c_i^* is applied to another model j. We show the comparison of applying c_i^* on model j with the best configuration c_j^*, and the results are shown in Fig. 2. Apparently, different DNNs require different configurations for better power efficiency. Due to space limitations, we exhibit the result for ResNet50 and WDSR in Fig. 2. When we apply the best-suited speed-oriented configuration of ResNet50 on WDSR, it can only achieve 42.12ms inference speed, while WDSR can deliver 36.14ms speed when using its best-suited configuration. Meanwhile, the best-suited configurations for WDSR could increase the power consumption when applied on ResNet-50 by 87.0% (from 1.62W to 3.03W) if compared with the best-suited configurations of ResNet-50 itself. These results put forward the necessity of DNN-adaptive CPU-GPU configuration.

Model	Input Size	Output Size	#Params	#FLOPs
ResNet50	$3 \times 224 \times 224$	1×1000	25.56M	8.22G
EfficientNet-b0	$3 \times 224 \times 224$	1×1000	5.27M	801.34M
WDSR	$3 \times 540 \times 960$	$3 \times 1080 \times 1920$	2.85K	2.80G

TABLE II: The input size, the total number of parameters, and total Floating-point operations (FLOPs) of each DNN model.

C. Huge Search Space of CPU-GPU Configuration

The heterogeneous CPU-GPU architecture in mobile devices provides rich selections in system parameter configurations, including three CPU core clusters, CPU, and GPU frequency. Take a recent OnePlus 8T as an example. It has 3 CPU core clusters, 8 CPU cores, 48 CPU frequencies, and 6 GPU frequencies, resulting in 367,289 available combinations. Collecting the performance for different DNN inference stages even expands the search space. Moreover, there are many prevailing DNN models with diverse resource demands, further complicating the DNN-adaptive CPU-GPU configuration task. As a result, it is impractical for designers to address the configuration issue manually.

Putting all together, we would raise a question: Is it possible to propose a novel framework to quickly achieve the Pareto-optimal latency and power consumption for various stages of different DNN inference tasks on mobile devices?

IV. Methodology

To address the issue above, we propose MOC, a Multi-Objective RL-assisted DNN inference-adaptive CPU-GPU Co-optimization framework. Fig. 3 shows the architecture of MOC, which is mainly composed of three components, including the demand-resource matching-based model, MODRL model, and rule-based action refinement technique.

A. DNN Demand-resource Matching Model

To facilitate more appropriate CPU-GPU configuration for different types of DNN models, we propose a heuristic model to classify the DNN algorithms according to their resource demands and the available hardware resources of the device. We introduce two models: 1) **FF (Feed-Forward) ceiling model**, and 2) **CM (CPU-Memory) ceiling model**.

FF ceiling model classifies DNNs for the feed-forward calculation stage, whose typical feature is that either GPU processors are waiting for data transmission or the memory

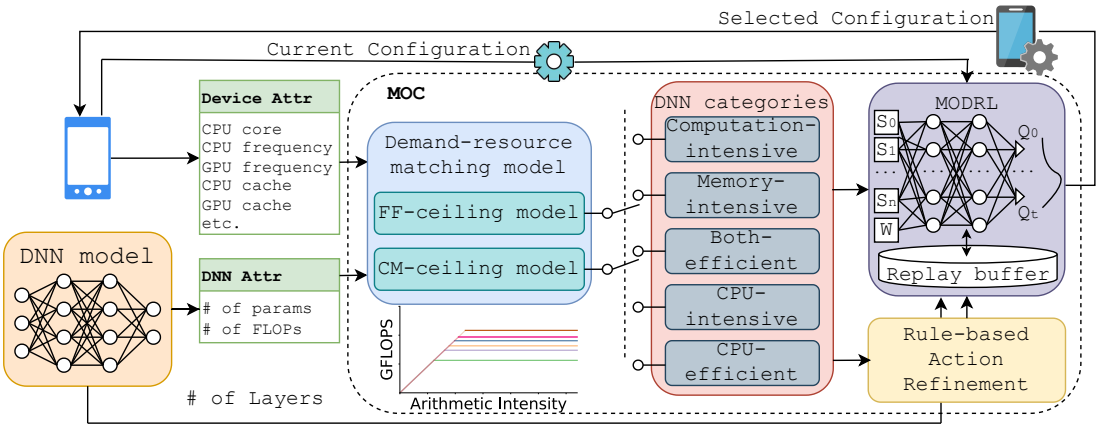


Fig. 3: Architecture of MOC. In MODRL, S_i is the i^{th} state, W is the weight vector, Q_t is the Q-value of t^{th}action.

system is waiting for the GPU calculation. The roofline model classifies DNNs by which operation is more intensive – data processing or data movement [34]. The current roofline models or their extension, e.g., [35], [36] neglect the frequent data reuse of DNN weights, which may lead to a misunderstanding of the relationship between GPU cache size and data memory footprint. We extend the roofline model by taking the frequent data reuse into consideration.

We first calculate the layer-wise cache-aware factor γ_ℓ by Eq. (1), where the $\lceil . \rceil$ is the ceil function, the GMEM_{cache} is the GPU cache size and ℓ indicates DNN layer ℓ. $\mathbf{Byte}_{\ell w}$ and $\mathbf{Byte}_{\ell i}$ are the weights and input feature map data size of layer ℓ, respectively. Note that $\mathbf{Byte}_{\ell i}$ for the CONV layer needs to be collected after implementing the image-to-column transform, since the CONV is implemented by GEMM. Then, we calculate the cache-aware arithmetic intensity $I_{a\gamma}$ by Eq. (2), where $\mathbb{L}$ is the number of layers, $\mathbf{FLOP}_\ell$ denotes layer ℓ's FLOP, $\mathbf{Byte}_\ell$ is the data size of layer ℓ, including the weights, feature map, and output, and γ_ℓ is from Eq. (1); Finally, the forward ceiling model can be obtained by Eq. (3), where k_{GPU} is a constant related to GPU, calculated by the number of processing units and the FLOPS per cycle of the processing unit, f_{GPU} is the GPU frequency, $k_{GPU} \times f_{GPU}$ approximates the calculation capacity of GPU with various GPU frequencies, and β is the DRAM bandwidth. The hint behind this is that the larger the DNN layer data size over the GPU cache size, the more requirements for the GPU to access the DRAM, compromising the effectiveness of the high DRAM bandwidth.

$$\gamma_\ell = \begin{cases} \left\lceil \frac{\mathbf{Byte}_{\ell i}}{\text{GMEM}_{cache} - \mathbf{Byte}_{\ell w}} \right\rceil & \text{GMEM}_{cache} > \mathbf{Byte}_{\ell w} \\ \left\lceil \frac{\mathbf{Byte}_{\ell i} + \mathbf{Byte}_{\ell w}}{\text{GMEM}_{cache}} \right\rceil & \text{GMEM}_{cache} < \mathbf{Byte}_{\ell w} \end{cases} \quad (1)$$

$$I_{a\gamma} = \frac{1}{|\mathbb{L}|} \sum_{\ell \in \mathbb{L}} \frac{\mathbf{FLOP}_\ell}{\mathbf{Byte}_\ell \times \gamma_\ell} \quad (2)$$

$$P_{peak} = \min \begin{cases} k_{GPU} \times f_{GPU} \\ \beta \times I_{a\gamma} \end{cases} \quad (3)$$

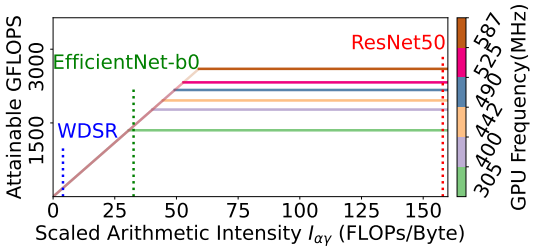


Fig. 4: FF ceiling model of OnePlus 8T.

Fig. 4 shows the FF ceiling model for OnePlus 8T. Multiple ceilings indicate the attainable performance is increasing with GPU frequency. Therefore, EfficientNet-b0 could be either memory-bound or computation-bound when GPU frequency varies. In contrast, WDSR is always memory-bound, while ResNet50 is always computation-bound.

CM ceiling model classifies DNN models in pre/post-process stages by comparing the waiting time of CPU cores and the memory system. To classify the DNN model as CPU-intensive or CPU-efficient, we introduce our CM ceiling model as Eq. (4), $\mathbf{Byte}_{data}$ is the data size copied to the CPU cache, $\text{CMEM}cache$ is the CPU cache size, which might be different for different CPU core clusters. This term ($t_{\text{PROCESSING}}$) defines the CPU time to read the input data from DRAM to the CPU cache. k_{CPU} is the constant related to the CPU architecture, and f_{CPU} is CPU frequency, which impacts data transmission and computing kernel initialization.

$$t_{\text{PROCESSING}} \approx \frac{\mathbf{Byte}_{data}}{\text{CMEM}_{cache}} \times k_{CPU} \times \frac{1}{f_{CPU}} \quad (4)$$

Since the input/output processing is mainly determined by the CPU, the processing time can be formulated by the number of copies required for each data and the speed of processing. Derived from the calculation of $u_{p.}$, the CM ceiling model can be illustrated as Fig. 5. The color intensity denotes the CPU frequency, the more intense, the higher frequency. The input data, with size $3 \times 224 \times 224$, occupies 558KB of memory and needs three copies to complete the transfer. For larger input/ output data size, such as WDSR in Table II, a input with size

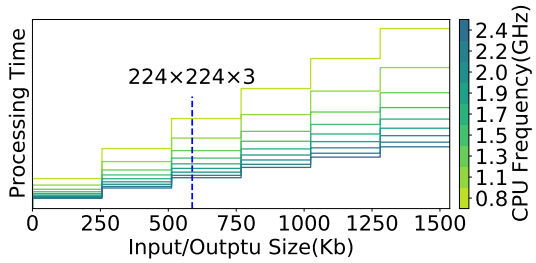


Fig. 5: Illustration of CM ceiling model for single medium CPU core of the OnePlus 8T ($Cache_{L2} = 256KB$), the color intensity denotes the CPU frequency.

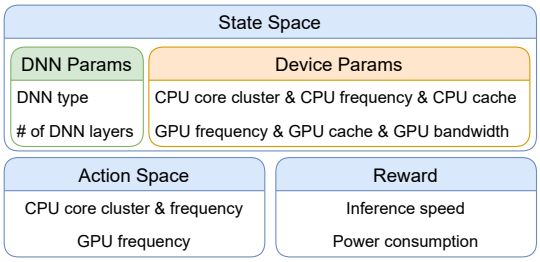


Fig. 6: Design space of MOC.

$3 \times 540 \times 960$ requires 5.93MB memory and at least 24 copies, not to mention the output with an even larger size.

According to the models above, we categorize DNN models into five categories, including three GPU-oriented categories (*computation-intensive*, *memory-intensive*, *both-efficient*) and two CPU-oriented categories (*CPU-intensive*, and *CPU-efficient*). A DNN model could fit one GPU-related category and one CPU-related category.

B. MODRL: Multi-objective Deep RL Model

We show the design space of MOC in Fig. 6 and describe each component of MOC in detail as follows.

1) State space: The state space considers both the DNN parameters and the mobile device running status. Unlike prior works [22], [23], MOC classifies the DNN models as a pre-knowledge to tailor the search space. Hence, we consider the DNN type and the DNN layer number in the state space. For the device parameters, we consider the current CPU-GPU configuration, as it could decide the current speed and power of the device. Moreover, we consider other hardware parameters, e.g., CPU cache, GPU cache, and GPU bandwidth. These parameters could vary from device to device, hence are taken into consideration to make MOC adaptive to given devices. Note that the DNN type is a two-dimensional tuple that contains the classification results of both FF and CM ceiling models. For pre/post-processing stages, the FF ceiling model DNN type is labeled as 0, as pre/post-processing stages are GPU-irrelevant. Similarly, in the feed-forward calculation stage, the DNN type of the CM ceiling model is labeled as 0.

2) Action space: MOC generates an action for each stage in DNN inference. Specifically, the action space consists of configuring the active CPU core cluster, CPU frequency, and GPU frequency. By traversing available combinations of the three parameters for each state in the training stage, MOC could search for the best-suited combination by finding the action with the maximum Q value in online decision-making.

3) Weight space: We take a weight vector as the input of MOC, as referring to current MORL algorithms [37]. Each weight in the vector denotes the relative importance between DNN inference speed and power consumption in the reward function. With our design, users can decide how many tuning policies to be used according to the requirements by simply specifying the length of the vector. For example, the user could configure the vector as $\{0,1\}$, in which 0 means power-oriented parameter tuning, while 1 means speed-oriented tuning. With the guidance of the weight vector, MOC could learn the tuning object of the user and perform fast convergence for all weights in a one-take training.

4) Reward: The reward in MOC is the weighted sum of normalized power and speed of each stage in DNN inference, formulated by Eq. (5), where α is the weight to determine the trade-off between the model speed and power consumption, and the speed and power are both normalized.

$$Re = \alpha \times (1 - speed) + (1 - \alpha) \times (1 - power) \quad (5)$$

5) Experience replay buffer: We adopt a multi-list formulated experience replay buffer in MOC. Each list in the buffer maintains all samples for each weight. By doing so, MOC only selects the sample from the same weight during training, thus avoiding interference from other weights and helping reduce the training effort to meet specific user objects.

6) Deep-Q-Network (DQN) model: A multi-head Multi-layer Perceptron (MLP) with three fully-connected layers combined with batch normalization and ReLU activation is adopted as the DQN model. The inputs are processed by the embedding layers that map the categorical or discrete input variables into continuous vector space. The multi-head denotes the DQN model contains multiple outputs corresponding to the candidate actions in the action space.

C. Rule-based Action Refinement

To further reduce the search space of MOC and boost its training process, we comprehensively studied the relationship between CPU-GPU configurations and DNN inference speed and power. Based on the results, several rules are proposed to refine the action space in MOC.

Rule 1: In the feed-forward calculation stage, MOC only selects actions with one little CPU core.

Feed-forward calculation stage mainly happens in the GPU, during which the CPU only takes charge of minor tasks, e.g., kernel activation, and task scheduling. Hence, one little CPU core is powerful enough to afford most DNNs in this stage.

Rule 2: In pre/post-processing stages, MOC only selects actions with the lowest GPU frequency.

Pre/post-processing stages mainly happen in CPU, hence we could choose the lowest GPU frequency in both stages.

Rule 3: In pre/post-processing stages, MOC should not choose the CPU core cluster with only one core.

Large amounts of data movements happen in the pre/post-processing stages, which demands enough CPU cache resources to boost both stages. As a result, one CPU core with only one CPU cache is insufficient for both stages.

The proposed rules are derived from a large number of evaluations conducted on multiple prevalent DNN models. Owning the space constraint, we exhibit representative data by separately examining the impacts of modifying the CPU frequency, the GPU frequency, and the number of CPU cores on the DNN inference stages of various DNN models using the OnePlus 8T. We decreased the CPU frequency from 1.8GHz to 0.6GHz and the GPU frequency from 587MHz to 305MHz. Additionally, we varied the number of CPU cores according to the order of reducing the total L2-cache size, which includes all CPU cores($1\times$ big core, $3\times$ mid cores, and $4times$ little cores), $3\times$mid cores, $4\times$little cores, $1\times$big core, $1\times$mid core, and $1\times$little core.

Fig. 7 shows the power consumption and latency of the feed-forward stage for ResNet50, EfficientNet-b0, and WDSR. Fig. 7 shows that, in the feed-forward stage, reducing the GPU frequency results in a decrease in power consumption; however, it also slows down the speed. Although the CPU frequency and the number of cores have minimal impact on the latency, the number of CPU cores has more influence on the power consumption reduction, which supports the rule of selecting actions with one little CPU core, thereby the *Rule 1*.

Fig. 8 shows the power consumption and latency of the pre/post-processing stage. We select the data size 1×1000 and $224\times224\times3$ that are usually used for image classification tasks, and $1080\times1920\times3$, widely used in tasks with HR image input/output. Notably, the GPU frequency has no impact on both the latency and power consumption since the GPU does not involve in the pre/post-processing stage. Hence it is intuitive to keep the GPU frequency to the lowest frequency to avoid the possible influence of the GPU, demonstrating the *Rule 2*. The number of CPU cores is more influential on latency compared to the CPU frequency. When reducing the number of CPU cores, the latency increases by factors of approximately $\times2.9$, $\times3.4$, and $\times9.2$ for data size 1×1000, $224\times224\times3$, and $1080\times1920\times3$, respectively. On the other hand, reducing the CPU frequency results in the inference latency increase of approximately $\times2.0$, $\times1.9$, and $\times1.2$ for the same data size. However, the number of CPU cores and the CPU frequency have a similar impact on reducing power consumption. It shows the important role of the CPU during pre/post-processing, thereby supporting the proposed *Rule 3*.

D. Overhead analysis

The overhead incurred by MOC is minor. The overhead of the Deep-Q Network model is 2.31ms per action on average. Besides, it only provides the best-suited configuration for each stage as preset. Thus it will not impact DNN inference. The primary overhead of MOC is DVFS switching, which only takes dozens of microseconds [25]. This overhead can be considered negligible compared to the latency of DNN inference stages.

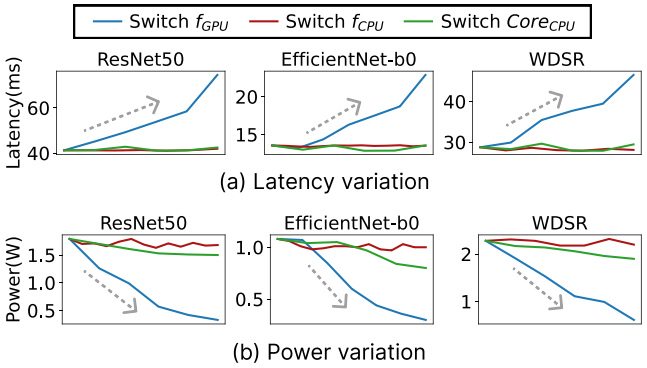


Fig. 7: Latency (a) and power (b) variation when switching the GPU frequency, CPU frequency, or CPU cores alone for ResNet50, WDSR, and EfficientNet-b0 during feed-forward stage on OnePlus 8T.

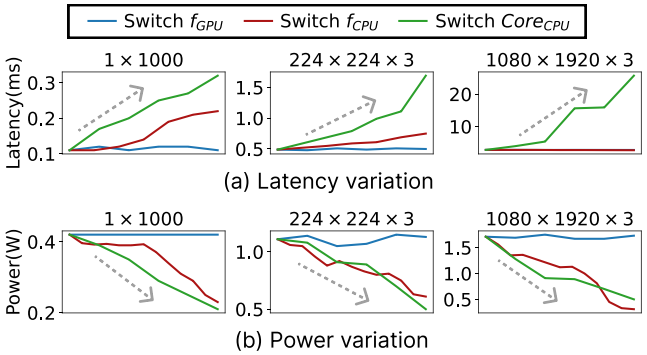


Fig. 8: Latency (a) and power (b) variation when switching the GPU frequency, CPU frequency, or CPU cores alone for 1×1000, $224\times224\times3$, and $1080\times1920\times3$ data size during the pre/post-processing stage on OnePlus 8T.

E. Implementation

MOC could be integrated as a lightweight component in the mobile compiler, i.e., CoCo-Gen [38] and TVM [39]. It contains pre-trained DQN models to provide suitable configurations for the model device. The API, `MOC.retrieve(DNN, `α`)`, is provided to the developer for retrieving the best-suited CPU-GPU configuration based on the current mobile configuration. Given the input `DNN` parameter, MOC could determine the corresponding DNN category with our DNN demand-resource matching model. α, as mentioned in Eq. (5), indicates the weight parameter between the speed latency and power consumption. Thus, the API retrieves the corresponding configuration based on the DNN model, the developer's demands, and the device specification for the DNN demand-resource matching model.

V. Evaluation

A. Experimental Environments

1) Experiemntal Setup: We use the OnePlus 8T [24] to perform the DNN inference. It has the Qualcomm SnapDragon 865 chipset with a Qualcomm Kyro 585 Cota-core CPU (1×2.84 GHz Cortex-A77 & 3×2.42 GHz Cortex-A77 & 4×1.80 GHz Cortex-A55) and a Qualcomm Adreno 650

CPU Cluster (#Cores)	f_{CPU} (GHz)	f_{GPU} (MHz)
little (4)	0.69, 0.78, 0.88, 0.97, 1.08, 1.17, 1.25, 1.34, 1.42, 1.52, 1.61, 1.71, 1.80	587
medium (3)	0.71, 0.83, 0.94, 1.06, 1.17, 1.29, 1.38, 1.48, 1.57, 1.67, 1.77, 1.86, 1.96, 2.05, 2.15, 2.25, 2.34, 2.42	525 490 441.6
big (1)	0.84, 0.96, 1.08, 1.19, 1.31, 1.4, 1.52, 1.63, 1.75, 1.86, 1.98, 2.07, 2.17, 2.27, 2.36, 2.46, 2.55, 2.65, 2.75, 2.84	400 305

TABLE III: Actions for the OnePlus 8T. The number of CPU cores in the cluster is shown after the cluster name.

GPU. Tab. III enumerates the actions of the OnePlus 8T. The Monsoon High Voltage Power Monitor (HVPM) is adopted to measure the power consumption of mobile devices [40].

2) MOC Training: The training codes are implemented on PyTorch. The DQN model is trained with collected data of ResNet50, EfficientNet-b0, and WDSR on a NVIDIA 2080Ti GPU. We train the model for 100k iterations using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.99$. The initial learning rate is set to 1×10^{-3} and halved at 50k and 75k iterations. The ϵ of ϵ-greedy strategy is initially set to 0.9 and decayed exponentially to 0.05. Various α weights are selected for multiple DQN models to compromise the different speeds or power consumption requirements.

3) DNN Inference: DNN inference is executed on OnePlus 8T with CoCo-Gen [38] as the compiler. Each testing DNN takes 1000 runs on different configurations. The average power consumption is obtained as the P_{avg} during each test. For a fair comparison, the power of the idle state is also collected as the baseline of the corresponding configuration. Then, $\Delta P = P_{avg} - P_{idle}$ is collected as the power consumption.

4) State-of-the-art (SOTA) works: MOC can be tuned by defining the α in reward to meet customized requirements. We compare MOC with **CDC**, **PowerSave**, **Schedutil**, and **TFlite**. Coarse-Grained Configuration (**CDC**) adopts DRL in coarse-grained DNN inference CPU-GPU configuration, where they neither consider the resource-demand for various models nor the different stages of DNN inference, with referring to [22], [23]. Due to no code being publicly released, we implemented the CDC methodology ourselves based on published papers. **PowerSave** [41] and **Schedutil** [42] are classic DVFS governors. Schedutil activates all CPU cores with varied CPU and GPU frequencies following the device resource utilization. PowerSave keeps CPU frequency to the user-defined lowest value. TFlite is a prevalent mobile DNN compiler [7], whose default configuration activates four little CPU cores with **Schedutil** DVFS governor. These SOTA works use one CPU-GPU configuration for the whole DNN inference process.

B. Experimental Results and Analysis

To demonstrate the advantages of MOC, we compare it with SOTA works on different DNN models. ResNet50 is a famous image classification with high computation resource demands. WDSR focuses on image super-resolution tasks that require huge memory and can be applied on the mobile device [7]. EfficientNet-b0 is an image classification model designed for efficient inference and has been widely applied to the model device. These three representative DNN models focus on different fields and have distinct resource demands.

We select different $\alpha \in \{0.1, 0.2, 0.3, ..., 1.0\}$ for the DQN model, where the α is the parameter in Eq. 5, to evaluate the performance of the MOC under different situations. For each α, we take the average inference speed and power consumption gathered from random initial states. The average inference speed and power consumption are taken as the performance of the corresponding DQN model. Fig. 9 and Tab. IV show the experimental results of various approaches. Several observations are induced from the results.

The CDC introduces DRL to optimize the device configuration to reduce power consumption but lacks to consider the different resource demands among each DNN inference stage and DNN categories. As shown in Tab. IV, we compare CDC using three different configurations, where CDC* and CDC† are configured to achieve a similar power and latency as MOC^2, respectively. MOC^2 can obtain faster inference speed (15.6% for ResNet50, 17.9% for WDSR, and 0.5% for EfficientNet-b0) compared to CDC*. In addition, MOC^2 can reduce power consumption by 24.8% for ResNet50, 11.4% for WDSR, and 34.6% for EfficientNet-b0 compared to CDC†. In addition, we also configure CDC$^\diamond$ to achieve a similar latency as MOC^3, which shows the capacity to retain an acceptable inference speed when ultra-low power consumption. Here MOC^3 mitigates power consumption by 52.3% for ResNet50, 20.8% for WDSR, and 56.8% for EfficientNet-b0 with almost 0.5% inference speed slower. This is because MOC performs a fine-grained CPU-GPU configuration for each stage in DNN inference, while CDC configures the CPU-GPU system for the whole DNN inference process.

Compared with the Schedutil, MOC could reduce power consumption by 67.2% for ResNet50, 45.1% for WDSR, and 75.5% for EfficientNet-b0, with minor defects on speed latency, 0.8% for ResNet50, 0.4% for WDSR, and 1.2% for EfficientNet-b0. Schedutil governor is designed for general tasks and only takes the processor's utilization to adapt the CPU frequency dynamically. Without considering the unique demands of DNN models fails to achieve a suitable performance for DNN applications.

The Powersave governor only reduces the CPU frequency. Although it could mitigate the power consumption incurred by the CPU, it will also induce inference speed degradation. MOC can outperform Powersave on ResNet50 (28.3% power reduction with only 9.8% slow in speed), WDSR (17.6% power reduction with even 5.7 % faster speed), and EfficientNet-b0 (34.7% power reduction with 9.3% faster speed). The reason is that activating suitable CPU cores can increase power efficiency while keeping fair inference speed with MOC.

TFlite mitigates the power issue of Schedutil by only activating the little CPU cluster, thereby improving power efficiency without significantly compromising inference speed. However, MOC can surpass TFlite by employing an adaptive CPU-GPU configuration that extends beyond the CPU cluster,

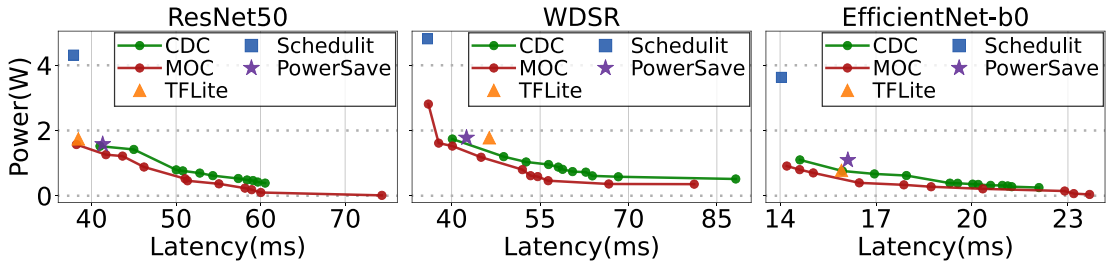


Fig. 9: Comparison of MOC with SOTA works.

Method	ResNet50		WDSR		EfficientNet-b0	
	T(ms)↓	ΔP(W)↓	T(ms)↓	ΔP(W)↓	T(ms)↓	ΔP(W)↓
Schedutil	37.88	4.32	35.89	4.81	14.04	3.63
PowerSave	41.32	1.52	42.51	1.88	16.12	1.09
TFlite	38.43	1.74	46.37	1.78	15.92	0.78
CDC*	54.31	1.08	48.82	1.53	15.95	0.74
CDC†	46.09	1.46	40.11	1.75	14.69	1.08
CDC$^\diamond$	58.16	0.44	56.80	0.77	19.36	0.37
MOC^1	**38.21**	**1.59**	**36.04**	**2.83**	**14.21**	**0.93**
MOC^2	**45.81**	**1.09**	**40.08**	**1.54**	**14.62**	**0.71**
MOC^3	**58.95**	**0.21**	**57.28**	**0.61**	**19.34**	**0.16**

TABLE IV: Comparison with SOTA methods. MOC^1, MOC^2, MOC^3 correspond to setting $\alpha = 0.9, 0.8, 0.2$ in reward function, respectively. CDC* is configured to achieve the **similar power** as MOC^2. CDC† is configured to achieve the **similar latency** as MOC^2. CDC$^\diamond$ is configured to achieve the **similar latency** as MOC^3.

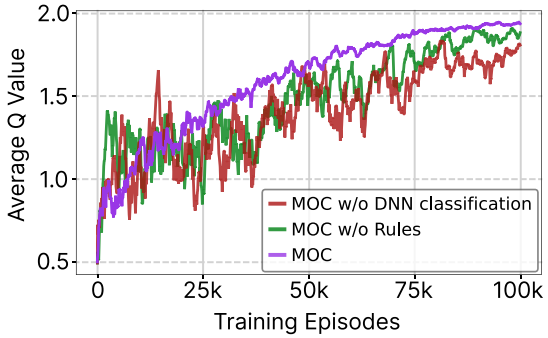


Fig. 10: Ablation study of MOC

allowing for enhanced performance and efficiency. MOC can reduce the power consumption by 18.9% for ResNet50, 39.2% for WDSR, and 4.5% for EfficientNet-b0), even speed up the inference time (0.5% for ResNet50, 13.6% for WDSR, and 6.3% for EfficientNet-b0).

MOC outperforms these works by both DNN-adaptive and fine-grained configurations. Moreover, MOC represents more flexibility over these works, as users could select their desired CPU-GPU configuring policy, e.g., power-oriented or speed-oriented, by setting the weight in the reward function in MOC.

C. Ablation Study

In the ablation study, we investigate the impact of our rule-based action refinement. By comparing MOC with and without rule-based action refinement during the deep reinforcement learning training process, it shows the advantages of action refinement in the MOC. Moreover, we also compared the influence of DNN model classification during each stage. The object is to demonstrate the MOC benefit from distinguishing DNN categories when providing suited configuration.

1) Importance of rule-based action refinement in MOC: The MOC could be revised without rule-based action refinement, where no actions will be ignored during the training. As shown in Fig. 10, the Q-value of MOC converges around 75k iterations while MOC without rule-based refinement converges around 90k iterations with a higher variation. This is because the rule-based action refinement provides prior knowledge that filters out those actions that might incur worse performance.

2) The necessity of DNN classification in MOC: The absence of DNN model classification in MOC leads to inferior Q-values in comparison to other approaches in Fig. 10. Various categories of DNN models exhibit distinct preferences for CPU-GPU configurations based on their CPU/GPU resource demands during different inference stages. Disregarding these attributes impedes MOC from obtaining suitable CPU-GPU configurations for each DNN model. These results verify the necessity of adopting DNN classification in MOC.

With both techniques, the oscillation of the Q-value of MOC can be effectively reduced, which leads to faster convergence and stable performance.

VI. Conclusion

This paper proposes MOC, a Multi-Objective CPU-GPU Co-optimization approach for power-efficient DNN inference on mobile devices. MOC strives to configure the CPU-GPU system adaptive to DNN inference stages, thus achieving a Pareto-optimal inference power and speed. First, a DNN demand-resource matching model is proposed to classify DNN models into 5 groups. Then, a multi-objective DRL model is established to adaptively configure the CPU-GPU system for each DNN inference stage. Finally, a rule-based action refinement technique is introduced to tailor the search space of MOC. Evaluations show that MOC could significantly reduce the power consumption of DNN inference while delivering an excellent speed. We expect that MOC could provide hints for subsequent works in DNN acceleration on mobile devices.

Acknowledgments. The research reported here was funded in whole/part by the Army Research Office/Army Research Laboratory via grant W911-NF-20-1-0167 to Northeastern University. Any errors and opinions are not those of the Army Research Office or Department of Defense and are attributable solely to the author(s). This research is also partially supported by NSF CCF-1937500 and CNS-1909172.

REFERENCES

[1] Y. Gong, Z. Zhan, P. Zhao, Y. Wu, C. Wu, C. Ding, W. Jiang, M. Qin, and Y. Wang, "All-in-one: A highly representative dnn pruning framework for edge devices with dynamic power management," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[2] W. Niu *et al.*, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, 2020.

[3] Y. Song *et al.*, "Dancing along battery: Enabling transformer with run-time reconfigurability on mobile devices," *DAC*, pp. 1003–1008, 2021.

[4] A. Goel, C. Tung, Y.-H. Lu, and G. K. Thiruvathukal, "A survey of methods for low-power deep learning and computer vision," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–6.

[5] P. Dong, S. Wang, W. Niu, C. Zhang, S. Lin, Z. Li, Y. Gong, B. Ren, X. Lin, and D. Tao, "Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[6] Z. Zhan, Y. Gong, P. Zhao, G. Yuan, W. Niu, Y. Wu, T. Zhang, M. Jayaweera, D. R. Kaeli, B. Ren, X. Lin, and Y. Wang, "Achieving on-mobile real-time super-resolution with neural architecture and pruning search," in *ICCV*, 2021.

[7] Y. Wu *et al.*, "Compiler-aware neural architecture search for on-mobile real-time super-resolution," in *ECCV*. Springer, 2022, pp. 92–111.

[8] G. Yuan, X. Ma, W. Niu, Z. Li, Z. Kong, N. Liu, Y. Gong, Z. Zhan, C. He, Q. Jin *et al.*, "Mest: Accurate and fast memory-economic sparse training framework on the edge," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 838–20 850, 2021.

[9] Y. Gong, Z. Zhan, Z. Li, W. Niu, X. Ma, W. Wang, B. Ren, C. Ding, X. Lin, X. Xu *et al.*, "A privacy-preserving-oriented dnn pruning and mobile acceleration framework," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 119–124.

[10] Z. Li, Y. Gong, X. Ma, S. Liu, M. Sun, Z. Zhan, Z. Kong, G. Yuan, and Y. Wang, "Ss-auto: A single-shot, automatic structured weight pruning framework of dnns with ultra-high efficiency," *arXiv preprint arXiv:2001.08839*, 2020.

[11] X. Ma, Z. Li, Y. Gong, T. Zhang, W. Niu, Z. Zhan, P. Zhao, J. Tang, X. Lin, B. Ren *et al.*, "Blk-rew: A unified block-based dnn pruning framework using reweighted regularization method," *arXiv preprint arXiv:2001.08357*, 2020.

[12] Z. Wang, Z. Zhan, Y. Gong, G. Yuan, W. Niu, T. Jian, B. Ren, S. Ioannidis, Y. Wang, and J. Dy, "Sparcl: Sparse continual learning on the edge," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 366–20 380, 2022.

[13] T. Jian, Y. Gong, Z. Zhan, R. Shi, N. Soltani, Z. Wang, J. Dy, K. Chowdhury, Y. Wang, and S. Ioannidis, "Radio frequency fingerprinting on the edge," *IEEE Transactions on Mobile Computing*, vol. 21, no. 11, pp. 4078–4093, 2021.

[14] Y. Gong, G. Yuan, Z. Zhan, W. Niu, Z. Li, P. Zhao, Y. Cai, S. Liu, B. Ren, X. Lin *et al.*, "Automatic mapping of the best-suited dnn pruning schemes for real-time mobile acceleration," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 5, pp. 1–26, 2022.

[15] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[16] J. Yu *et al.*, "Wide activation for efficient and accurate image super-resolution," *arXiv preprint arXiv:1808.08718*, 2018.

[17] M. Halpern *et al.*, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in *HPCA*, 2016, pp. 64–76.

[18] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated cpu-gpu power management for 3d mobile games," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[19] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*. IEEE, 2007, pp. 28–38.

[20] big.LITTLE, "Arm big.little heterogeneous processing architecture." https://www.arm.com/technologies/big-little.

[21] S. M. Nabavinejad *et al.*, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2496–2508, 2022.

[22] M. Han *et al.*, "Herti: A reinforcement learning-augmented system for efficient real-time inference on heterogeneous embedded systems," in *PACT*. IEEE, 2021, pp. 90–102.

[23] Y. G. Kim *et al.*, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," in *MICRO*, 2020, pp. 1082–1096.

[24] Oneplus, "Oneplus 8t." https://www.oneplus.com/global/8t/.

[25] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, 2013.

[26] V. Peluso *et al.*, "Performance profiling of embedded convnets under thermal-aware dvfs," *Electronics*, vol. 8, no. 12, p. 1423, 2019.

[27] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 598–610. [Online]. Available: https://doi.org/10.1145/2830772.2830797

[28] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and thermal management in the intel core duo processor." *Intel Technology Journal*, vol. 10, no. 2, 2006.

[29] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: Association for Computing Machinery, 2009, p. 469–480. [Online]. Available: https://doi.org/10.1145/1669112.1669172

[30] S. M. Nabavinejad *et al.*, "Coordinated dvfs and precision control for deep neural networks," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 136–140, 2019.

[31] K. Rungsuptaweekoon, V. Visoottiviseth, and R. Takano, "Evaluating the power efficiency of deep learning inference on embedded gpu systems," in *2017 2nd International Conference on Information Technology (INCIT)*. IEEE, 2017, pp. 1–5.

[32] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 487–498. [Online]. Available: https://doi.org/10.1145/2485922.2485964

[33] M. Tan *et al.*, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*. PMLR, 2019, pp. 6105–6114.

[34] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[35] Y. Wang *et al.*, "Time-based roofline for deep learning performance analysis," in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*. IEEE, 2020, pp. 10–19.

[36] M. Hill and V. Janapa Reddi, "Gables: A roofline model for mobile socs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 317–330.

[37] A. Abels *et al.*, "Dynamic weights in multi-objective deep reinforcement learning," in *ICML*. PMLR, 2019, pp. 11–20.

[38] S. Liu *et al.*, "Cocopie: Making mobile ai sweet as pie–compression-compilation co-design goes a long way," *arXiv preprint arXiv:2003.06700*, 2020.

[39] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze *et al.*, "Tvm: an automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation*, 2018, pp. 579–594.

[40] M. S. Inc., "Moonson high performance power monitor." https://www.msoon.com/online-store.

[41] K.-D. Kang, G. Park, H. Kim, M. Alian, N. S. Kim, and D. Kim, "Nmap: Power management based on network packet processing mode transition for latency-critical workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 143–154. [Online]. Available: https://doi.org/10.1145/3466752.3480098

[42] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, *AsyMo: Scalable and Efficient Deep-Learning Inference on Asymmetric Mobile CPUs*.

New York, NY, USA: Association for Computing Machinery, 2021, p. 215–228. [Online]. Available: https://doi.org/10.1145/3447993.3448625