DVFO: Learning-Based DVFS for Energy-Efficient Edge-Cloud Collaborative Inference

Ziyang Zhang, *Student Member, IEEE*, Yang Zhao, *Senior Member, IEEE*, Huan Li, *Senior Member, IEEE*, Changyao Lin, *Student Member, IEEE*, and Jie Liu, *Fellow, IEEE*

***Abstract*—Due to limited resources on edge and different characteristics of deep neural network (DNN) models, it is a big challenge to optimize DNN inference performance in terms of energy consumption and end-to-end latency. In addition to dynamic voltage frequency scaling (DVFS) technique, edge-cloud architecture provides a collaborative approach for efficient DNN inference. However, current edge-cloud collaborative inference methods have not optimized various compute resources on edge devices. Thus, we propose DVFO, a novel DVFS-enabled edge-cloud collaborative inference framework, which co-optimizes DVFS and offloading parameters via deep reinforcement learning (DRL). Specifically, DVFO automatically co-optimizes 1) the CPU, GPU and memory frequencies of edge devices, and 2) the offloaded feature map. In addition, it leverages a *thinking-while-moving* concurrent mechanism to accelerate the DRL learning process, and a *spatial-channel attention* mechanism to identify the less important DNN feature map for efficient offloading. This approach improves inference performance for different DNN models under various edge-cloud network conditions. Extensive evaluations using two datasets and six widely-deployed DNN models on five heterogeneous edge devices show that DVFO significantly reduces the energy consumption by 33% on average, compared to state-of-the-art schemes. Moreover, DVFO achieves up to 28.6%∼59.1% end-to-end latency reduction, while maintaining accuracy within 1% loss on average.**

***Index Terms*—Edge computing, DVFS technology, collaborative inference, deep reinforcement learning.**

I. Introduction

As THE development of edge computing [1] and lightweight deep learning techniques, edge devices equipped with Internet of Things (IoT) connectivity and hardware accelerators (e.g., GPUs) are becoming capable of executing deep neural network (DNN) in real-time for many edge intelligence [2] applications, such as defect detection [3], face recognition [4], and mobile augmented reality [5], just to name a few. For instance, autonomous driving [6] requires running latency-critical DNN inference tasks to achieve both high real-time performance and satisfactory quality of service (QoS) [7]. However, compared to cloud servers, edge devices have fewer compute resources and more stringent power consumption requirements, thus it is more challenging to optimize DNN inference performance in terms of energy consumption and end-to-end latency on local edge devices.

To achieve efficient DNN inference on resource-constrained edge devices, it is a promising approach to reduces the end-to-end latency or energy consumption of edge devices via various techniques such as dynamic voltage frequency scaling (DVFS) [9], [10], and edge-cloud collaborative inference [11], [12]. DVFS is a low-power technology that dynamically adjusts the voltage and frequency according to energy consumption. Prior work [13] has proposed a series of deep reinforcement learning-based DVFS techniques to reduce energy consumption. However, DVFS reduces energy consumption by increasing end-to-end latency, which we illustrate and discuss in Section II. In addition, none of the existing methods above considers the edge-cloud collaboration paradigm. The edge-cloud collaborative inference offloads partial DNN feature map from edge devices to cloud servers, with edge devices inferring partial DNN, cloud servers executing the rest, and small neural networks to fuse them to obtain the final inference results [14]. To avoid network bottlenecks to achieve offloading DNN feature map efficiently, prior work utilizes explainable AI [12] and compressed sensing [14] to compress feature map. However, the expensive runtime overhead of these schemes still impairs DNN inference real-time performance.

Combining DVFS and edge-cloud collaboration, prior work [15] proposes a data offloading scheme, namely DRLDO, which uses deep reinforcement learning together with DVFS to reduce energy consumption. However, DRLDO only considers CPU core voltage and frequency in DVFS, without including the GPU and memory resources. In addition, it does not consider performance bottlenecks of various DNN models. Recent benchmarks reveal that GPUs are responsible for around 70% of the total energy consumption during DNN training [16]. As shown in Fig. 1, we perform experiments and show that during DNN inference phase, GPUs also consume more energy than CPUs for all the DNN models that we have investigated. We report the normalized energy usage of different compute units including CPU, GPU, and memory, when executing four DNN models with CIFAR-100 [8] dataset on an NVIDIA Xavier NX edge device. The result shows that the energy consumption

Manuscript received 26 June 2023; revised 17 January 2024; accepted 17 January 2024. Date of publication 23 January 2024; date of current version 3 September 2024. This work was supported in part by the National Key R&D Program of China under Grants 2021ZD0110905 and 2022YFF0503900, and in part by An Open Competition Project of Heilongjiang Province, China, on Research and Application of Key Technologies for Intelligent Farming Decision Platform, under Grant 2021ZXJ05A03. Recommended for acceptance by E. Hossain. *(Corresponding authors: Yang Zhao; Jie Liu.)*

Ziyang Zhang and Changyao Lin are with the School of Science and Technology, Harbin Institute of Technology, Harbin 150006, China (e-mail: zhangzy@stu.hit.edu.cn; lincy@stu.hit.edu.cn).

Yang Zhao, Huan Li, and Jie Liu are with the International Research Institute for Artificial Intelligence, Harbin Institute of Technology, Shenzhen 518071, China (e-mail: yang.zhao@hit.edu.cn; huanli@hit.edu.cn; jieliu@hit.edu.cn).

Digital Object Identifier 10.1109/TMC.2024.3357218

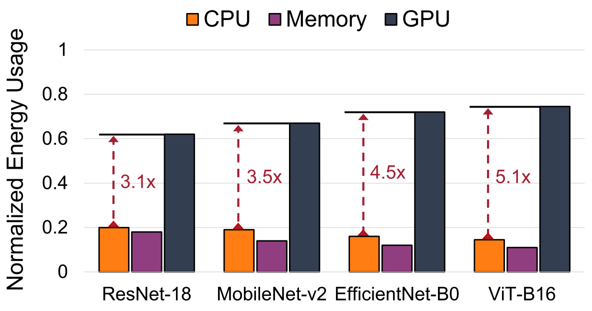

Fig. 1. Energy usage of CPU, GPU and memory for four DNN inference models with CIFAR-100 [8] dataset, measured on NVIDIA Xavier NX. We set the batch size to 1.

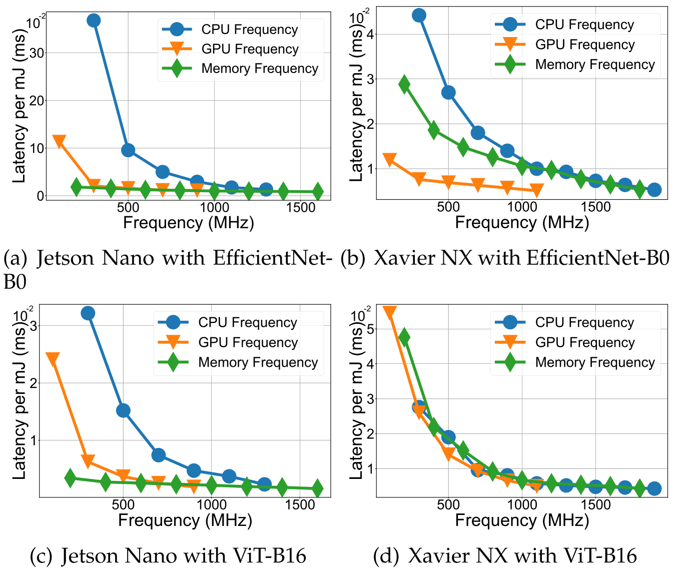

Fig. 2. Inference performance (i.e., *latency per mJ*) of five heterogeneous edge devices with different CPU, GPU and memory frequencies for EfficientNet-B0 [25] and Visual Transformer (ViT-B16) [26] DNN models under CIFAR-100 [8] dataset. We set the batch size to 1.

of the GPU is 3.1× to 3.5× that of the CPU, indicating that GPU dominates DNN inference. It can also be observed that since DNN inference accesses memory frequently, the energy consumption of the memory is not negligible. In addition, as shown in Fig. 2, the performance of different DNN models has diminishing returns as hardware frequencies increase. Learning DNN model behaviors on different edge devices can further improve inference performance and energy efficiency. All these observations motivate us to incorporate CPU, GPU and memory resources in DVFS, and utilize feature map offloading for DNN inference on edge devices.

Table I provides a comparison of key features of DVFO with four dimensions of DVFO to related work, including DVFS technology and edge-cloud collaborative inference. DVFS technology enables on-device DNN inference with lower energy consumption. While DRLDO [15], CARTAD [17] and QL-HDS [18] have achieved energy-efficient inference on multi-core CPU systems using DVFS technology, they did not consider edge devices with CPU-GPU heterogeneous processors, which are crucial for GPU-dominated energy-efficient on-device inference. DeepCOD [14] and AgileNN [12] compressed the offloaded DNN feature map, but the compression overhead is not negligible. Since most of the works mentioned above do not combine DVFS with edge-cloud collaborative inference, in this paper we showcase how to achieve low latency and energy consumption using learning-based DVFS in an edge-cloud collaboration framework.

TABLE I
Comparison of Key Features of DVFO With Prior Work

Service Framework	Enable DVFS	Collaborative Inference	Data Compression	Enable GPU device
DRLDO [15]	✓	✓	✗	✗
CARTAD [17]	✓	✗	✗	✗
QL-HDS [18]	✓	✗	✗	✗
AppealNet [19]	✗	✓	✗	✓
DeepCOD [14]	✗	✓	✓	✓
AgileNN [12]	✗	✓	✓	✓
GearDVFS [20]	✓	✗	✗	✓
DVFO (Ours)	✓	✓	✓	✓

In order to achieve energy-efficient DNN inference, in this paper, we propose DVFO, a DVFS enabled learning-based collaborative inference framework that automatically co-optimizes the CPU, GPU and memory frequencies of edge devices, as well as the DNN feature map to be offloaded to cloud servers. We need to deal with the following issues to design and implement such a framework. First, edge-cloud collaborative inference has dynamic network conditions and intense real-time requirements. Deep reinforcement learning (DRL) is effective in dealing with high-dimensional decision and optimization problems, but existing methods applied to edge-cloud collaboration are inefficient to deal with the real-world dynamic environment, e.g., online policy inference cannot catch dynamic environment changes [21]. Thus, we utilize a concurrency mechanism, called *thinking-while-moving* [22], to accelerate policy inference for agents in DRL, as we discuss in details in Section V-A. Second, the feature map to be offloaded to cloud servers would have a network bottleneck, which can dramatically increase transmission latency and energy consumption. We leverage a *spatial-channel attention* mechanism [23] to guide feature map offloading [12], so that the end-to-end latency can be significantly reduced without sacrificing DNN inference accuracy.

After solving these issues, we perform experiments and compare DVFO with state-of-the-art methods on CIFAR-100 [8] and ImageNet-2012 [24] datasets. Extensive evaluations show that DVFO can efficiently balance energy consumption and end-to-end latency by automatically co-optimizing the hardware resources of edge devices and the feature map to be offloaded to cloud servers.

In summary, we make the following contributions:

- We propose DVFO, a novel DVFS enabled edge-cloud collaborative DNN inference framework that automatically co-optimizes the hardware frequencies of edge devices, and the proportion of the feature map to be offloaded to cloud servers.
- We apply the *thinking-while-moving* concurrent control mechanism in learning-based optimization, and we design an importance-based feature map offloading scheme to alleviate edge-cloud network bottlenecks by leveraging a *spatial-channel attention* mechanism.

- Extensive evaluations on five heterogeneous edge devices with two datasets show that DVFO reduces energy consumption by up to 33% on average for various DNN models, compared to state-of-the-art schemes. DVFO also achieves 28.6%∼59.1% end-to-end latency reduction, without scarifying accuracy.

The rest of the article is organized as follows: Section II highlights our research motivations. Section III briefly describes deep reinforcement learning we used. Section IV describes system overview and problem formulation. Section V illustrates our framework design in detail. Section VI reports experimental results. Section VII presents related work. Section VIII concludes our work.

II. Motivation

Although DNN models can provide state-of-the-art performance for many IoT applications, it comes at the cost of intensive complexity and prohibitive energy consumption. Therefore, it is critical to be able to efficiently execute DNN on resource-constrained edge devices. In this section, we discuss the experiments and observations that motivate us to develop an efficient DVFS enabled learning-based edge-cloud collaborative inference framework.

As mentioned in Section I, we perform experiments with two widely-deployed DNN models (i.e., EfficientNet-B0 [25] and ViT-B16 [26]), and observe that GPU consumes more energy than CPU during the DNN inference phase on edge devices. To better understand the impact of CPU, GPU and memory frequencies of edge devices on the end-to-end latency and energy consumption, we further conduct the following experiments and analysis in Fig. 2. As you can see, we execute memory-intensive DNN model (e.g., EfficientNet-B0 [25]) and compute-intensive (e.g., ViT-B16 [26]) DNN model [27] on an NVIDIA Jetson Nano and NVIDIA Xavier NX edge platform, respectively.

Note that prior work only considers end-to-end latency or energy consumption as a single metric, which cannot directly reveal the trade-off between inference performance and energy requirements. We report the inference performance *latency per mJ*, a metric by dividing end-to-end latency by energy consumption. As shown in Fig. 2, we measure the inference performance of two heterogeneous edge devices with two aforementioned DNN models under CIFAR-100 [8] dataset using different CPU, GPU and memory frequencies. We have the following key observations from our experiments and analysis:

- *High frequency does not mean high inference performance:* Intuitively, the higher frequency is, the larger amounts of energy the system consumes. However, increasing frequency does not improve inference performance (i.e., *latency per mJ*). Take EfficientNet-B0 [25] as an example, the energy consumption with the maximum frequency doubled after 500 MHz, but the end-to-end latency is not significantly reduced, which means that the inference performance tends to saturate. Similar phenomenon can be observed for Vision Transformer (ViT-B16) [26]. Therefore, a learning approach is needed to automatically find the appropriate hardware frequencies to achieve optimal inference performance.
- *DNN models with different operation intensities reveal significant end-to-end latency and energy differences on heterogeneous edge devices:* Take for example the NVIDIA Xavier NX edge platform, which has abundant compute resources. According to operational density in the roofline model [27], we can conclude from the Fig. 2(b) that EfficientNet-B0 [25] is a memory-intensive DNN, because the performance bottleneck depends on the CPU and memory frequencies. The ViT-B16 [26] with higher complexity in Fig. 2(d) is a compute-intensive DNN model, where GPU frequency dominates performance. However, these two DNN models are both compute-intensive on Jetson Nano, which has limited compute resources compared with Xavier NX. Thus, it illustrates that the same DNN model exhibit high heterogeneity for edge devices with different computing resources, and DVFS alone cannot further improve inference performance. Therefore, we highlight that identifying the behavior of various DNN models under heterogeneous devices can further improve inference performance.

In addition to the observations mentioned above, we also point out the need to adjust feature map offloading adaptively based on edge-cloud network conditions for various IoT and mobile computing applications. Take autonomous driving for example. More data can be offloaded to cloud, when driving in urban areas with high quality wireless network connection, while offloading needs to be reduced for rural areas with limited network bandwidth. Thus, it motivates us to learn the feature map offloading proportion parameter for various network conditions. As shown in Section VI-D, our DVFO framework can take advantage of the abundant resources on cloud servers to offload more feature map for high network bandwidth conditions, while our algorithm learns to use different offload proportion parameters for poor network conditions.

To summarize, we highlight two schemes that can achieve optimal trade-off between energy consumption and end-to-end latency for energy-efficient DNN inference: (1) dynamic voltage and frequency scaling (DVFS), and (2) edge-cloud collaborative inference. Note that the above two schemes are orthogonal. For instance, DVFS can reduce energy consumption by adjusting hardware frequency, but it increases end-to-end latency. Edge-cloud collaborative inference can effectively reduce end-to-end latency by offloading data while further reducing energy consumption on edge devices.

III. Preliminaries

Deep reinforcement learning (DRL) combines deep learning and reinforcement learning, where reinforcement learning is used to define problems and optimize objectives, and deep learning is used to solve the modeling of policy and value function (V-function) in reinforcement learning. In general, DRL uses the back-propagation algorithm to optimize the objective function, which is suitable for solving complex high-dimensional sequential decision problems and achieves impressive performance on many tasks. The agent in DRL is used to perceive the environment and make decisions, which performs a task by interacting with the external environment. Meanwhile, the environment

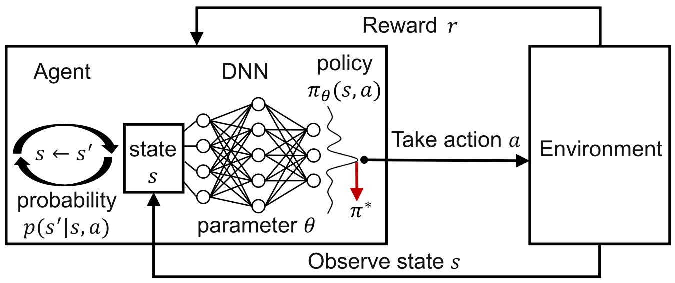

Fig. 3. Overview of deep reinforcement learning system.

changes its state by responding to the actions selected by the agent, and feeds back corresponding reward signals to the agent.

As shown in Fig. 3, most DRL algorithms take the optimization problem as a markov decision process (MDP), which can be described by a tuple: $(\mathcal{S}, \mathcal{A}, \pi, r, p)$, where $\mathcal{S}$ is the state space containing all states $s(s \in \mathcal{S})$; $\mathcal{A}$ is the action space containing all actions $a(a \in \mathcal{A})$; π is the probability distribution function that determines the next action a according to the state s, satisfying $\sum_{a\in\mathcal{A}} \pi(a|s) = 1$; r is a scalar function, which means that after the agent makes an action a according to the current state s, the environment feeds back a reward signal to the agent. Note that r is related to the state s' at the next moment due to hysteresis; p is the state transition probability, which means that after the agent makes an action a according to the current state s, the probability that the environment changes to the state s' at the next moment, also satisfying $\sum_{s'\in\mathcal{S}} p(s'|s,a) = 1$.

The goal of the DRL algorithm is to find an optimal policy π^* to maximize the following expected return:

$$\pi^* = \underset{\theta}{\text{argmax}}\, \mathbb{E}_{\tau\sim p(\tau)} \left[\sum_{t=0}^{T-1} \gamma^{t-1} r_t \right], \quad (1)$$

where $\tau = s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}$ is a trajectory that represents an interaction process between the agent and the environment. θ is the parameter of policy network, and $\gamma \in [0, 1]$ is a discount factor. We can obtain the optimal policy $\pi^* = \text{argmax}_a\, Q^*(s, a)$ by value iteration via the following the Bellman optimal equation of state-action value function (Q-function):

$$Q^*(s,a) = \mathbb{E}_{\tau\sim p(\tau)}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})]. \quad (2)$$

In Section V-A, we describe the DQN-based DRL algorithm in detail.

IV. System Overview and Problem Statement

A. System Overview

Fig. 4 shows an overview of our DVFO framework. The framework incorporates local DNN inference on edge devices and remote DNN inference on cloud servers. During DNN inference, users submit DNN inference tasks to DVFO, along with user-defined parameters to adjust the trade-off between energy consumption and end-to-end latency (i.e., the weight parameter η in (4)), and the workflow starts as follows. ❶ DVFO utilizes a feature extractor on edge devices to extract high-dimensional features of the input data and obtain DNN feature map. The feature extractor is implemented based on a lightweight neural network with negligible overhead. ❷ To alleviate network bottlenecks of the feature map to be offloaded to cloud servers, DVFO utilizes *spatial-channel attention* module to evaluate the importance of feature map, in order to guide the feature map offloading. The attention module details are in Section V-B. ❸ The DRL-based DVFO module (i.e., DVFO optimizer) learns the optimal hardware frequency vector and the proportion parameter of the feature map to be offloaded to cloud servers for each task based on historical data, current bandwidth, and user configuration (see Section V-A for more details). ❹ Based on the optimal hardware frequencies and the feature map to be offloaded to cloud servers learned by DVFO optimizer, DVFO retains the top-k features with high importance for local DNN inference, and then combines the remote DNN with other compressed less important features via weighted summation (the summation weight parameter $\lambda \in (0, 1)$ can also be user-defined), to produce the final prediction result on edge devices locally. Compared to adding additional neural network (NN) layers for fusion, such a point-to-point weighted summation method is much more lightweight and has low computation overhead on edge [12].

B. Problem Statement

Opportunities to reduce the energy consumption of DNN inference come at the cost of increased end-to-end latency. When optimized for energy consumption, DNN end-to-end latency (i.e., time-to-inference, or TTI) may be impaired. Here we define the energy consumption of DNN inference as its energy-to-inference (ETI)

$$\text{ETI}(\mathbf{f}, \xi) = \text{TTI}(\mathbf{f}, \xi) \times \text{AvgPower}(\mathbf{f}, \xi), \quad (3)$$

where $\mathbf{f}$ and ξ are the hardware frequency vector of device, and the proportion of the feature map to be offloaded to cloud servers, respectively, and AvgPower is the average power consumption during inference with configuration $(\mathbf{f}, \xi)$. Different from prior work [15] that only considers the CPU frequency f^C, we also incorporate GPU and memory frequencies of edge devices, denoted as f^G, f^M, respectively, that is, $\mathbf{f} = (f^C, f^G, f^M)$.

Cost Metric: It is important to define a cost metric in designing DVFO, so that users can adjust the trade-off between energy consumption and end-to-end latency based on the application requirements and their preferences. Thus we propose the following cost metric:

$$C(\mathbf{f}, \xi; \eta) = \eta \cdot \text{ETI}(\mathbf{f}, \xi) + (1 - \eta) \cdot \text{MaxPower} \cdot \text{TTI}(\mathbf{f}, \xi), \quad (4)$$

where MaxPower is the maximum power limit supported by device, a constant introduced to unify the units of measure in the cost metric [28], and $\eta \in [0, 1]$ is a weight parameter that users define to adjust the balance between energy consumption and end-to-end latency. In particular, when $\eta = 0$, we are only optimizing energy consumption ETI, whereas when $\eta = 1$, only end-to-end latency TTI is optimized. A more detailed sensitivity analysis of the parameter η can be found in Section VI.

End-to-End Latency Model: For a set of DNN inference tasks $\mathcal{X} = (x_1, x_2, \ldots, x_N)$ consisting of N independent and non-preemptive tasks x_i, $i = 1, \ldots, N$. We show the optimization

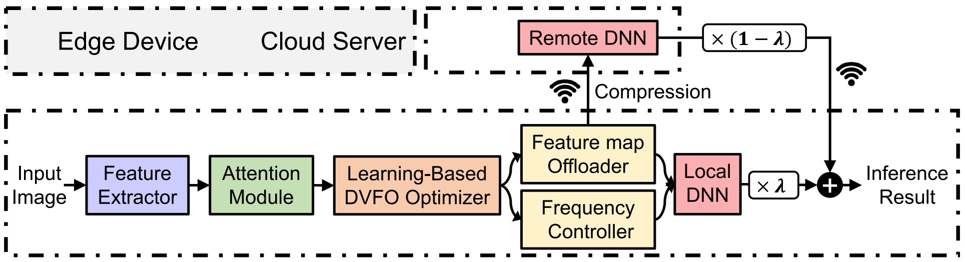

Fig. 4. Overview of the architecture of DVFO framework.

problem in terms of end-to-end latency and energy consumption. First, for end-to-end latency TTI_i^{total}, it incorporates 1) the computing time on edge for the ith task TTI_i^{local}, 2) the compression (quantization) time of the feature map to be offloaded to cloud servers on edge TTI_i^{comp}, 3) the transmission time of the offloaded feature map to cloud TTI_i^{off}, and 4) the computing time on cloud TTI_i^{cloud}. Note that we ignore the fusion time on edge devices and the decompression time on cloud servers, benefit from the lightweight weighted summation-based fusion method on edge devices in Section V-C and the abundant computing resources of the cloud servers, respectively.

To be more specific, the computing time on edge TTI_i^{local} depends on two factors: the size of feature map without offloading m_i^{local}, and the hardware frequency of edge devices $(f_{local}^C, f_{local}^G, f_{local}^M)$, which can be defined as

$$\mathrm{TTI}_i^{local} = \frac{m_i^{local}}{(f_{local}^C, f_{local}^G, f_{local}^M)}, \tag{5}$$

Unlike the conventional tasks oriented in prior work [29], DVFO focuses on inference tasks based on DNN models, that is, the DNN inference performance depends not only on the CPU frequency (for data preprocessing), but is also affected by the GPU frequency (for parallel computation) as well as the memory frequency (for data copying between the CPU and GPU).

Likewise, the computing time on cloud TTI_i^{cloud} depends on the size of the feature map to be offloaded to cloud servers m_i^{cloud}, and the hardware frequency of cloud servers $(f_{cloud}^C, f_{cloud}^G, f_{cloud}^M)$

$$\mathrm{TTI}_i^{cloud} = \frac{m_i^{cloud}}{(f_{cloud}^C, f_{cloud}^G, f_{cloud}^M)}, \tag{6}$$

The compression time on edge TTI_i^{comp} depends on the size of the feature map to be offloaded to cloud servers m_i^{cloud}. In this work, we use quantization aware training (QAT) in Section VI-A to effectively compress the offloaded feature map with low-bit quantization (i.e., converted from float-32 model to int-8 model). The compression time on edge TTI_i^{comp} defined as

$$\mathrm{TTI}_i^{comp} = \mathrm{QAT}(m_i^{cloud}), \tag{7}$$

The transmission time TTI_i^{off} is affected by the size of the feature map to be offloaded to cloud servers m_i^{cloud} and the communication bandwidth $\mathcal{B}$, that is

$$\mathrm{TTI}_i^{off} = \frac{m_i^{cloud}}{\mathcal{B}}, \tag{8}$$

Note that the size of the feature map to be offloaded to cloud servers m_i^{cloud} is determined by the proportion parameter ξ in (4).

Therefore, the end-to-end latency TTI_i^{total} can be formulated as follows:

$$\mathrm{TTI}_i^{total} = \mathrm{TTI}_i^{local} + \mathrm{TTI}_i^{comp} + \mathrm{TTI}_i^{off} + \mathrm{TTI}_i^{cloud}. \tag{9}$$

Energy Consumption Model: For energy consumption, the overall energy consumption ETI_i^{total} of edge devices for a particular task x_i consists of the energy consumption for computing ETI_i^c and the energy consumption for offloading ETI_i^o, that is

$$\mathrm{ETI}_i^{total} = \mathrm{ETI}_i^c + \mathrm{ETI}_i^o. \tag{10}$$

To be more specific, the energy consumption for computing ETI_i^c of ith task x_i depends on the edge computing time TTI_i^{local} and the power consumption of edge devices P_i, which can be defined as

$$\mathrm{ETI}_i^c = \mathrm{TTI}_i^{local} \cdot P_i, \tag{11}$$

where the power consumption P_i of the edge device is the summation of dynamic P_i^D, static P_i^S, and constant P_i^C power consumption [30]. P_i^D originates from the activity of the logic gates inside the processor, which can be derived by $P_i^D = CV^2(f^C, f^G, f^M)$. Where C is the capacitance of the switching logic gate. P_i^S originates from transistor static current when the processor is powered on, which is described by $P_i^S = VN_{tr}I_S$. N_tr is the number of logic gates, and I_S is the normalized static current of each logic gate. P_i^C is the power consumption of peripheral devices, such as board fans and peripheral circuitry.

The energy consumption of offloading ETI_i^o for x_i is affected by the communication bandwidth $\mathcal{B}$ of the network between edge devices and cloud servers, the proportion of the feature map to be offloaded to cloud servers m_i^{cloud}, and the power consumption of edge devices P_i, that is

$$\mathrm{ETI}_i^o = \frac{m_i^{cloud} \cdot P_i}{\mathcal{B}}. \tag{12}$$

The objective of of DVFO is to minimize the cost in (4) by automatically exploring the feasible set of edge hardware frequency vector (f^C, f^G, f^M) and the offloading proportion parameter ξ. $\mathbf{f}_{\min}$ and $\mathbf{f}_{\max}$ ensures that hardware frequency vector (f^C, f^G, f^M) is not beyond its feasible change range $[\mathbf{f}_{\min}, \mathbf{f}_{\min}]$ specified in the hardware manual. Put formally in terms of the cost function defined by (4), our objective becomes

$$(\mathbf{P}) : \min_{\mathbf{f},\xi} .C(\mathbf{f}, \xi; \eta)$$

TABLE II
NOTATION AND DESCRIPTION

Notation	Description
$\mathcal{X}$	the whole task set
x_i	the i-th non-preemptive task
TTI	the time-to-inference
ETI	the energy-to-inference
C	the cost metric
f^C	the CPU frequencies of edge devices
f^G	the GPU frequencies of edge devices
f^M	the memory frequencies of edge devices
ξ	the proportion of the feature map to be offloaded
η	the weight parameter
m_i^{local}	the size of feature map without offloading
m_i^{cloud}	the size of feature map with offloading
$\mathcal{B}$	the communication bandwidth
α	the TOPS metric of edge devices
P_i	the power consumption of edge devices
λ	the summation weight local and remote inference results

$$\text{s.t. } \mathbf{f}_{\min} \leq (f^C, f^G, f^M) \leq \mathbf{f}_{\max}$$
$$0 \leq \xi \leq 1. \tag{13}$$

For each task, DVFO can automatically co-optimize CPU, GPU and memory frequencies, as well as the proportion of the feature map to be offloaded to cloud servers. Note that we assume cloud servers have enough compute resources to guarantee the real-time performance of remote inference. We also assume that edge devices can be put into idle mode after the inference and offloading operations to save energy.

Complexity Analysis: To represent the search space in DVFO, let C be the number of available CPU clock frequencies, G be the number of available GPU clock frequencies, and M be the number of available memory clock frequencies. Therefore, there are total $C \times G \times M$ possible options to pick the processor clock frequencies. Since the objective of DVFO is to co-optimize the clock frequencies of the processors as well as the proportion of feature map to be offloaded, the complexity of search space is $O(CGM\xi)$.

It is non-trivial to search exhaustively or greedily for the optimal joint configuration in polynomial time. To address this problem, we leverage a learning-based approach to effectively reduce the search space by concurrency control. Deep reinforcement learning (DRL), as a semi-supervised method, which does not require manually labeling data, is suitable for decision-making problems in complex environment, especially for large-scale state spaces and dynamically changing environment, where traditional supervised and unsupervised learning methods are usually powerless. The agent in DRL can learn the optimal policies by interacting with the environment, and does not need a prior knowledge. On this basis, DRL is capable of autonomous learning and incremental optimization in complex environment to address more complex and realistic problems. In addition, DRL is adaptive, able to adjust its policies and actions in real time, in response to changes in the environment and feedback rewards. These properties make it well suited to the dynamic environment of our problem.

Table II provides the notation and corresponding descriptions used in this paper.

V. System Design

A. Learning-Based DVFO

In this section, we describe how DVFO determines the hardware frequency vector $\mathbf{f}$ and the proportion of feature map to be offloaded ξ for each task. Inspired by recent learning-based DVFS methods [20], [31], the problem $\mathbf{P}$ in (13), can be converted to a reinforcement learning (RL) problem [32] to solve. We first formulate the optimization problem as a markov decision process (MDP), and utilize deep reinforcement learning (DRL) to automatically determine the optimal configuration.

To be more specific, the agent in DRL has three components, namely state, action and reward, which are defined as follows:

- *State Space:* At each time step t, the agent in DRL will construct a state space $\mathcal{S}$. We define the weight parameter η specified by the user, the summation weight parameter λ, the importance distribution of features $\mathbf{x} \sim \mathbf{p}(\mathbf{a})$, and the current network bandwidth $\mathcal{B}$ as state. The above measures constitute the state space $\mathcal{S}$, denoted as $\mathcal{S} = \{\lambda, \eta, \mathbf{x} \sim \mathbf{p}(\mathbf{a}), \mathcal{B}\}$.
- *Action Space:* We set the frequency vector $\mathbf{f}_i$ and the offloading proportion parameter ξ_i for x_i as actions. Therefore, the action space can be expressed as $\mathcal{A} = \{\mathbf{f}_i, \xi_i\}$, where $\mathbf{f}_i = (f_i^C, f_i^G, f_i^M)$ represents the CPU, GPU and memory frequencies for a particular task x_i. For example $(1500, 900, 1200, 0.3)$ means that 30% of feature map are executed locally, and the remaining of the feature map are offloaded to the remote, when the CPU, GPU and memory frequencies are set to 1500 MHz, 900 MHz and 1200 MHz, respectively.
- *Reward:* We propose a novel reward function for DVFO, which introduces the trade-off between energy consumption and end-to-end latency. To handle the constraints of (4), the clock frequency range as well as the offloading proportion of feature map are added to the reward function R. Therefore, the reward function in DRL is defined as follows:

$$R = \begin{cases} \frac{C(\mathbf{f}, \xi; \eta)}{e^{-\mathbf{f}/\xi}}, & \mathbf{f} \in [\mathbf{f}_{\min}, \mathbf{f}_{\min}] \\ \rho, & \text{otherwise} \end{cases}, \tag{14}$$

  where $\xi \in [0, 1]$ is the proportion of the feature map to be offloaded to cloud servers. In the first case, the clock frequency is within $[\mathbf{f}_{\min}, \mathbf{f}_{\min}]$, and we calculate the reward based on the cost in (4). Note that we introduce the parameters $\mathbf{f}$ and ξ, with the purpose of giving more reward when the clock frequency of edge device is higher and the proportion of feature map is lower. Otherwise, less rewards will be given. The second case represents that the current frequency exceeds the specified frequency range, meaning that the target cannot be achieved even if the minimum or maximum clock frequency is used. In other words, using the minimum or maximum clock frequency is actually the "optimal" action. Therefore, a small negative reward ρ $(= -0.1$ experience) is given with penalty.

However, as Fig. 5 shows, most DRL algorithms assume that the state of the environment is static, in which the agent is making a decision. That is, the agent first observes the state and then executes policy inference. However, this blocking

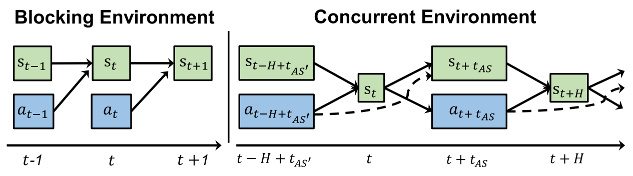

Fig. 5. Action trajectories in blocking and concurrent environment.

approach of sequential execution is not suitable for real-world dynamic real-time environment. Because the state has "slipped" after the agent observes the state of the environment and executes an action, i.e., the previous state transitions to a new unobserved state. This environment is regarded as a concurrent environment in [22]. In particular, in the edge-cloud collaboration environment with strict time constraints, we need to use DRL to adjust the frequency of edge devices and the proportion of feature map to be offloaded in real-time, according to the importance of features and network bandwidth. Therefore, it is crucial to reduce the overhead of policy inference in DRL. Furthermore, unlike zTT [31] which considers only a subset of the action space at a time, although this approach reduces computational complexity, agent fails to fully explore the environment resulting in sub-optimal performance. Note that DVFO can provide optimal solution for a particular edge device.

To overcome these issues, in this work, we utilize DQN [33] to learn the optimal CPU, GPU and memory frequencies, as well as the proportion of feature map to be offloaded for edge devices. We use the concurrency control mechanism to reduce the overhead of policy inference in DQN with continuous-time based on a *thinking-while-moving* mechanism [22]. The right part of Fig. 5 illustrates this concurrent approach. Specifically, the agent observes the state of the environment s_i at time step t. When it selects an action $a_{t+t_{AS}}$, the previous action $a_{t-H+t_{AS'}}$ has slid to a new unobserved state $s_{t+t_{AS}}$, meaning that state capture and policy inference in concurrent environment can be executed concurrently. Here H is the duration of the action trajectory from the state s_t to s_{t+t_H}.

We implement policy inference in concurrent environment by modifying standard DQN. The continuous-time Q-value function for the concurrent case from (2) can be expressed as following:

$$Q^{\pi}(s(t), a_{t-1}, a_t, t, t_{AS}) = \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{t-1}(t'))\, dt' + \gamma^{t_{AS}} \max_{a_{t+1}} \mathbb{E}_p Q^{\pi}(s(t+t_{AS}), a_t, a_{t+1}, t+t_{AS}, H-t_{AS}). \tag{15}$$

where t_{AS} is the time duration of state capture, policy inference. The Lemma 3.1. in [22] proved that *the concurrent continuous-time Bellman operator is a contraction*, which means that the concurrent continuous-time Q-function can eventually converge to the optimal Q-value Q^* in (2). In this way, agent can explore in parallel the clock frequency of each processor as well as the proportion of the feature map to be offloaded. Since the number of actions per branch is small, all possible values for each domain can be explored. On the other hand, the thinking-while-moving concurrency control mechanism learns the joint features of all domains for fine-grained parameter tuning. By doing so, the complexity of the search space can be reduced from $O(CGM\xi)$ to $O(C+G+M+\xi)$. In DVFO, we employ a *thinking-while-moving* mechanism to realize this concurrency control.

Algorithm 1: DVFO Optimization Process.

```
Input : user preference $\lambda, \eta$; feature map importance $\mathbf{x} \sim \mathbf{p}(\mathbf{a})$, and current network bandwidth $\mathcal{B}$
Output: the optimal settings of hardware frequency $\mathbf{f}_i$ and offloaded proportion $\xi_i$ for each task $x_i$
1  Initialize the parameters of network $Q$ and target network $Q'$ with $\theta_1$ and $\theta_2$, respectively;
2  Initialize an empty replay memory $\mathcal{D} \leftarrow \varnothing$;
3  Observe state $s_0 = \{\lambda, \eta, \mathbf{x} \sim \mathbf{p}(\mathbf{a}), \mathcal{B}\}$;
4  Initialize action $a_0 = \{\mathbf{f}_0, \xi_0\}$ randomly;
5  for environment step $t \leftarrow 1$ to $T$ do
6      for the $i$-th stage $i \leftarrow 1$ to $N$ do
7          Observe state $s_t$ in concurrent environment;
8          Select an action $a_t$ using thinking-while-moving with $\epsilon$-greedy;
9          Feed frequency controller and feature map offloader, respectively;
10         Execute computing and offloading, while obtaining reward $r$ using Eq. (14);
11         Set $s_t \leftarrow s_{t+1}$;
12         Store transition $(s_t, a_t, r(s_t, a_t), s_{t+1})$ in $\mathcal{D}$;
13     end
14 end
15 for gradient step $g \leftarrow 1$ to $G$ do
16     Sample minibatch of transitions form $\mathcal{D}$;
17     Calculate Q-value using Eq. (15);
18     Update $\theta_1$ via gradient descent;
19 end
```

Algorithm 1 illustrates the optimization process of DVFO in detail. We first initialize the parameters of neural network and replay memory in DRL. Then we take $\{\lambda, \eta, \mathbf{x} \sim \mathbf{p}(\mathbf{a}), \mathcal{B}\}$ as the initial state. At the start of training, the agent in DRL will select an action randomly. In each time step t, the agent captures the state s_t in a continuous-time concurrent environment, and chooses an action a_t using a *thinking-while-moving* concurrency mechanism. We use the ϵ-greedy strategy to explore the environment. Next, we feed the CPU, GPU, and memory frequencies, as well as the proportion of feature map to be offloaded, selected by the agent to frequency controller and feature map offloader, respectively. Simultaneously, the agent obtains an instant reward r, and the state changes from s_t to s_{t+1}. We store the current state, action, reward, and the state of the next time step as a transition in the replay memory. At each gradient step, we first sample mini-batch transitions from replay memory randomly. Then we use (15) to calculate the Q-value in the concurrent environment and update the network parameters using gradient descent. Finally, we deploy the trained DVFO online to evaluate the performance. Note that the training process is offline.

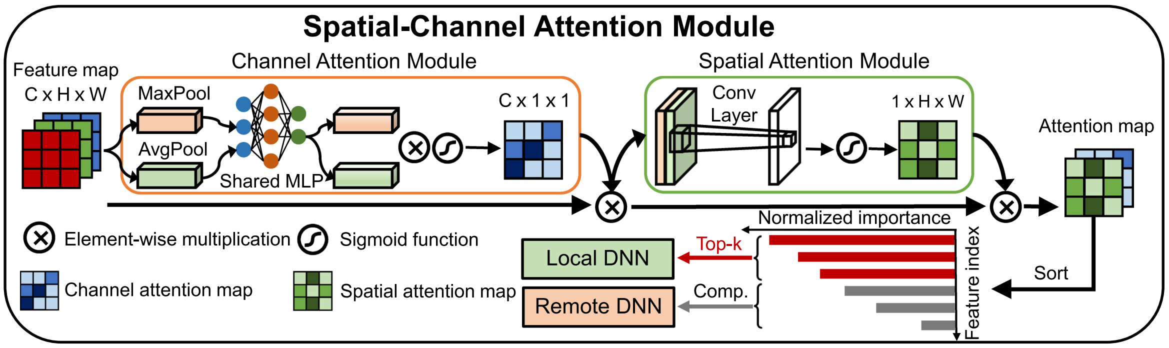

Fig. 6. Overview of *spatial-channel attention* module (SCAM). The module has two sequential sub-modules: *channel attention* module and *spatial attention* module. The intermediate feature map are divided into the top-k features with high importance and the remaining less important features by SCAM, which are executed by local DNN and remote DNN, respectively.

B. Spatial-Channel Attention Module

The feature map in DNN [12] is the data computed in a layer of a deep neural network (DNN). Specifically, given input data (e.g., an image), the feature map of a DNN is computed by utilizing a specific transformation (e.g., convolution, pooling, activation function, etc.) on a particular layer of the neural network. This transformation is determined by the weights and biases of the neural network, and these weights and biases can be learned during training. The goal of feature map is to extract useful features of the input data so that the neural network can utilize them for tasks (e.g., classification, regression, etc.). Note that feature map is closely related to DNN. In convolutional neural networks (CNN), feature map is critical because is intuitively reveals how the neural network extracts features from the input data. For instance, at early layers of a CNN, feature map shows that the neural network extracting edges or color patches of an image, while at deeper levels, feature map might show that the network extracting more complex features, such as faces or vehicles.

In practice, we leverage existing deep learning frameworks to determine the feature map of unknown DNN through the following procedures: 1) forward propagation: the input data is forward propagated via neural network, which involves a series of weight matrices, biases and activation functions. 2) selection of layers: a particular layer of the feature map is selected, which can be any convolutional layer, fully connected layer, or other type of layer. 3) extracting activations: for the selected layer, the activations corresponding to the input data, i.e., the feature map, is extracted. 4) visualization: use explainable tools to visualize the extracted feature map. Note that the above procedures may vary depending on the specific deep learning framework (such as TensorFlow, PyTorch, etc.) or specific network type (such as CNN, RNN, Transformer, etc.).

Our experiments show that the Pearson correlation coefficient between the importance-based indexing and inference contributions of each layer in the DNN is -0.738, which means there is a linear negative correlation between these two variables. Furthermore, we also propose a neural network-based spatial-channel attention module in Fig. 6 to reveal this correlation more intuitively.

The effectiveness of offloading in DVFO depends on the skewness [12] of the importance distribution of feature map. The higher the skewness, the fewer features dominate DNN inference. Therefore, we leverage a *spatial-channel attention* mechanism, namely *spatial-channel attention* module (SCAM) as shown in Fig. 6, to evaluate the feature importance of input data. Attention is a widely used deep learning technique that allows a network to focus on relevant parts of the input, and suppress irrelevant ones. We use it to identify the high important features and the remaining less important features for guiding feature map offloading.

In this way, we can reduce transmission latency by offloading the compressed less important features without significantly sacrificing the accuracy of DNN models. Note that our proposed SCAM is transparent to the DNN's architecture. In other words, once the feature map of a DNN has been determined using existing deep learning frameworks, SCAM is able to evaluate the importance of feature map without priori knowledge. Therefore, SCAM is applicable to DNNs with any architecture.

Given a feature map $\mathbf{F} \in \mathbb{R}^{C\times H\times W}$ extracted by feature extractor as input, SCAM sequentially infers a 1D channel attention map $\mathbf{M_c} \in \mathbb{R}^{C\times 1\times 1}$ and a 2D spatial attention map $\mathbf{M_s} \in \mathbb{R}^{1\times H\times W}$. For the arrangement of sequential process, experimental results in [23] show that channel-first is better than spatial-first. We next describe the details of each module.

1) Channel Attention Module: In general, since each channel of a feature map in DNN is considered as a feature detector, the channel attention module in SCAM focuses on "what" is meaningful given an input data. To fully extract richer channel attention, we aggregate the spatial information of the feature map using average pooling $(\mathrm{AvgPool})$ and max pooling $(\mathrm{MaxPool})$. We then feed the generated average-pooled features and max-pooled features into a shared network consisting of multi-layer perceptron (MLP) to obtain channel attention map. The channel attention is computed as follows:

$$\mathbf{M_c}(\mathbf{F}) = \sigma(\mathrm{MLP}(\mathrm{AvgPool}(\mathbf{F})) + \mathrm{MLP}(\mathrm{MaxPool}(\mathbf{F}))), \tag{16}$$

where σ denotes the sigmoid function.

2) Spatial Attention Module: As a complement to *channel attention*, *spatial attention* focuses on "where" is an informative part. We also use average pooling and max pooling along the channel axis to aggregate spatial information of feature map. The generated average pooling features and max pooling features

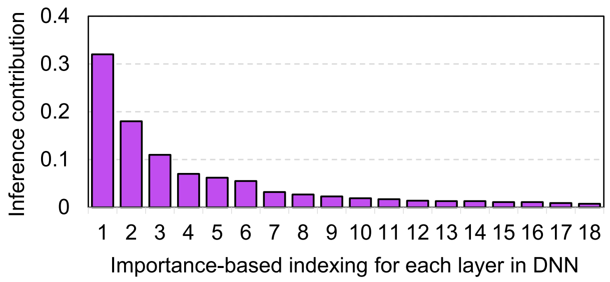

Fig. 7. Inference contribution of each layer in ResNet-18 [34] to CIFAR-100 [8] dataset (descending order).

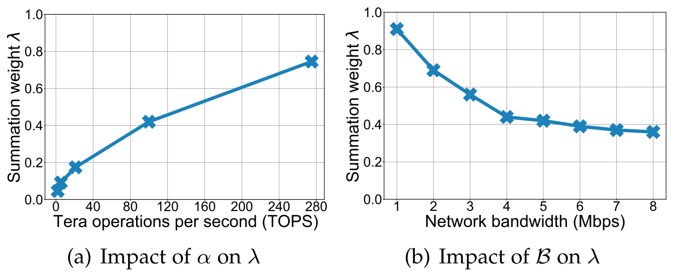

Fig. 8. Impact of (a) TOPS metric α and (b) network bandwidth $\mathcal{B}$ on the summation weight parameter λ. We use EfficientNet-B0 DNN model.

are concatenated and convolved by a 3×3 convolutional layer to generate a spatial attention map. The spatial attention is computed as follows:

$$\mathbf{M_s}(\mathbf{F}) = \sigma\left(Conv(3,3)[\mathrm{AvgPool}(\mathbf{F}); \mathrm{MaxPool}(\mathbf{F})]\right), \tag{17}$$

where $Conv(3,3)$ represents a convolution operation with a filter size of 3×3.

Arrangement of Attention Modules: Based on the channel attention map and spatial attention map obtained by (16) and (17), we can obtain the final attention map $\mathbf{F}^{out}$ by element-wise multiplication.

$$\mathbf{F}^{in} = \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F},$$
$$\mathbf{F}^{out} = \mathbf{M_s}\left(\mathbf{F}^{in}\right) \otimes \mathbf{F}^{in}, \tag{18}$$

where $\otimes$ denotes element-wise multiplication, $\mathbf{F}^{in}$ is the intermediate attention map. We can derive the importance distribution of features $\mathbf{x} \sim \mathbf{p}(\mathbf{a})$ from the normalized weights in final attention map $\mathbf{F}^{out}$, where $\mathbf{x}$ represents the feature map index, and $\mathbf{a} \in (0,1)$ is the normalized feature importance.

Fig. 7 illustrates the descending inference contribution of each layer in ResNet-18 [34] for CIFAR-100 [8] dataset, which evaluated by SCAM. Intuitively, only a few features make major contributions to DNN inference (e.g., top-3 features with the highest importance dominate 60% of contribution for the whole DNN feature map), while a large number of remaining less important features contribute insignificantly to DNN inference. In this way, we can evaluate the importance of different features and keep the top-k features with high importance for edge execution, while the remaining less important features are compressed, and then offloaded for remote execution. Compared with other explainable AI (XAI) approaches (e.g., CAM [35], Grad-CAM [36], etc.), SCAM is a lightweight and general module that can be integrated into DNN architecture with negligible overhead and trained end-to-end together with DNN models.

In addition, offloading the less important feature map is also a challenge especially with low edge-cloud network bandwidth. Inspired by SPINN [11], we introduce precision quantization (i.e., convert the feature map with 32-bit floating-point numbers to 8-bit fixed-length numbers) that compress the less important feature map to further reduce transmission latency. In this way, DVFO can effectively reduce the size of the less important feature map without significant information loss.

C. Combining Local and Remote Inference Results

DVFO leverages a *spatial-channel attention* mechanism to infer the high important feature map on edge devices, while cloud servers infer the remaining less important feature map. In order to efficiently and accurately fuse the inference results of both edge devices and cloud servers, DVFO applies weighted summation to fuse the inference results, and produces the final inference output at edge devices locally.

However, it is indeed a challenge to adaptively adjust the summation weight parameter used in the DVFO framework. To address this issue, we introduce the *tera operations per second* (TOPS) performance metric α, an include both effects of the TOPS metric and network bandwidth $\mathcal{B}$ in the determination of the summation weight parameter λ. We perform experiments and collect data on five different edge devices with different network bandwidth conditions to collect data for training the summation weight parameter. In addition, DVFO can further incorporate techniques, such as few-shot learning [37] and transfer learning [38], to learn DNN parameters more efficiently, when no sufficient data is available.

We leverage a two-layer lightweight convolutional neural network with 16 neural network units per layer, to optimize the summation weight parameter λ using stochastic gradient descent (SGD) algorithm [39], and deploy the trained λ in DVFO. By doing so, DVFO can flexibly adjust the weights of each component based on the TOPS metric of the edge device and changes in network bandwidth to optimize the overall performance. We evaluate in detail the effect of weighted summation on accuracy and energy consumption in Section VI-E.

As shown in Fig. 8(a), the summation weight parameter λ is proportional to the TOPS metric α of local edge device. Obviously, the higher the TOPS metric of edge device, the larger the contribution of local inference results. Fig. 8(b) reports the effect of network bandwidth on the summation weight parameter. It can be concluded that the higher $\mathcal{B}$, the smaller α, which means that the weight of local inference results decreases with the improvement of network bandwidth, and on the contrary, remote inference dominates the combination results.

Weighted summation in DVFO we used has the following advantages, compared to neural network-based prior work such as adding an extra convolutional layer for fusion [14]. First, the inference outputs of edge devices and cloud servers always maintain the same dimension. In contrast, using neural network (NN) layers (e.g., a fully connected or convolutional layer) to fuse these two outputs could possibly break such data alignment,

TABLE III
SPECIFIC PARAMETERS OF THE EDGE-CLOUD COLLABORATION DEVICE

	Platform	TOPS Metric	CPU	GPU	DRAM	Power
Edge	NVIDIA Jetson Nano	1.9TOPS (INT8)	4×Cortex-A57@1.4GHz	128×Maxwell@0.9GHz	4GB	5W-10W
	NVIDIA Jetson TX2	5.34TOPS (INT8)	6×Cortex-A57@1.4GHz	256×Pascal@1.3GHz	8GB	7.5W-15W
	NVIDIA Xavier NX	21TOPS (INT8)	6×Carmel@1.4GHz	384×Volta@1.1GHz	8GB	10W-20W
	NVIDIA Orin NX	100TOPS (INT8)	8×Cortex-A78AE@1.4GHz	1024×Ampere@0.9GHz	16GB	10W-20W
	NVIDIA AGX Orin	275TOPS (INT8)	12×Cortex-A78AE@2.2GHz	2048×Ampere@1.3GHz	64GB	15W-60W
Cloud	NVIDIA RTX 3080	476TOPS (INT8)	16×Intel Xeon Gold 6226R@2.9GHz	8740×Ampere@1.4GHz	128GB	320W

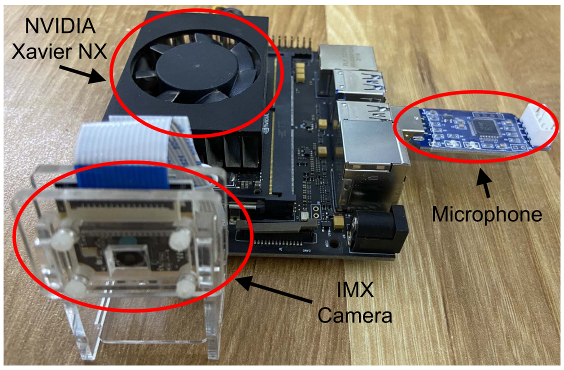

Fig. 9. DVFO prototype implementation deployed on NVIDIA Xavier NX edge platform for edge inference services. We use an IMX camera and a micorphone as IoT devices.

hence reducing the accuracy of the final inference. Second, such lightweight point-to-point weighted summation has less computation than neural networks, and adds negligible overhead relative to the inference at edge devices locally.

VI. PERFORMANCE EVALUATION

A. DVFO Implementation

As shown in Fig. 9, we use PyTorch 1.8 to implement DVFO on an edge device, i.e., NVIDIA Xavier NX for inference service as a case study. An IMX camera and a microphone are used as IoT device to generate image and speech data. Our proposed Algorithm 1 in learning-based DVFO is trained offline with four NVIDIA GeForce GTX 3080 GPUs, and we convert the local DNN from a float-32 model into an int-8 model using quantization aware training (QAT) supported by PyTorch. Different from post training dynamic quantization (PTDQ) and post training static quantization (PTSQ), QAT turns on the quantization function during the training process. Since quantization essentially converts the high precision of the model into low precision, which is likely to cause model performance degradation. In this case, QAT is better than PTDQ and PTSQ. In addition, both the network and target network with the prioritized experience replay and ϵ-greedy policy in DRL are trained using Adam optimizer. Each network has three hidden layers and one output layer, and each hidden layer has 128, 64, and 32 neural network units, respectively. We set the learning rate, buffer size and minibatch to 10^{-4}, 10^{6} and 256, respectively.

Table III lists specific parameters of edge devices and cloud servers used in DVFO. DVFO controls the CPU and GPU clock frequencies of edge devices whose specifications are shown in Table III. For instance, the CPU of Jetson NX has a clock frequency range of 0.1 GHz to 1.4 GHz, and the GPU clock frequency range of 0.1 GHz to 1.1 GHz. Due to communication overhead limitations, we select only one edge device and one remote server for edge-cloud cooperative inference in our experimental evaluation. Intuitively, the remote servers used in DVFO could also be replaced edge devices with high performance. Furthermore, we use nvpmodel, a power management tool from NVIDIA, which support flexible hardware frequency scaling on-device.

B. Experiment Setup

Datasets and DNN Models: We evaluate DVFO on CIFAR-100 [8] and ImageNet-2012 [24] datasets, respectively. The images with different sizes can comprehensively reflect the diversity of input data. Due to limited compute resources on edge devices, we set the batch size to be one for edge-cloud collaborative inference. We successfully integrated DVFO with state-of-the-art DNN models and thoroughly evaluated DVFO in the following three typical applications: object detection, image classification, and speech recognition. Apart from EfficientNet-B0 and ViT-B16, we additionally use six DNN models in Table V from three popular DNN families to process image and speech data, as these DNN models comprise most of edge applications. Moreover, the remote DNN in DVFO is constructed by removing the first convolutional layer from the benchmark DNN [12].

Energy Consumption Measurement: As described in Section IV-B, the overall energy consumption of edge devices incorporates computing and offloading energy consumption. To be more specific, we use jetson-stats [40], an open source monitoring toolkit to periodically profile and record the overall energy consumption of edge devices in real time.

Baselines: We compare DVFO with the following four approaches. Note that all experimental results are averaged over the entire test dataset.

- *AppealNet [19]:* An edge-cloud collaborative framework that decides whether the task uses a lightweight DNN model on edge devices or a complex DNN model on cloud servers by identifying the difficulty of the input data.
- *DRLDO [15]:* A DVFS-aware offloading framework that automatically co-optimizes the CPU frequency of edge devices and the offloaded input data.
- *Cloud-only:* The whole feature map are offloaded to cloud servers without edge-cloud collaboration inference.
- *Edge-only:* The whole model is executed on edge devices without edge-cloud collaboration inference.

Since AppealNet deploys DNN with different complexity at edge devices and cloud servers, respectively, we use the same DNN, including DVFO all the time, in order to make fair

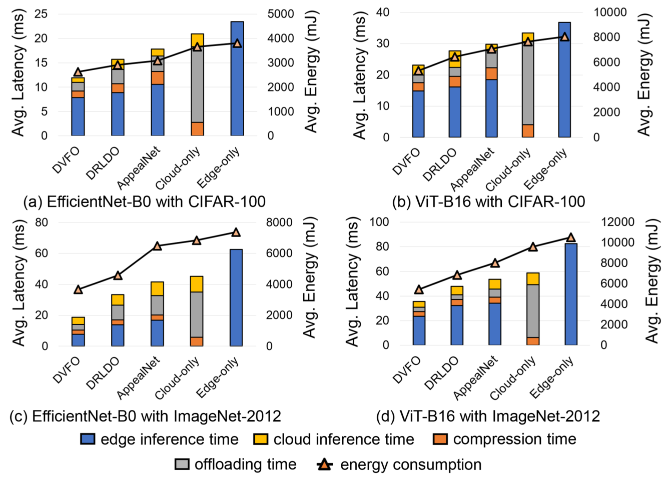

Fig. 10. Comparison of end-to-end latency and energy consumption for EfficientNet-B0 and Vision Transformer using Edge-only and edge-cloud collaborative inference on CIFAR-100 [8] and ImageNet-2012 [24] datasets.

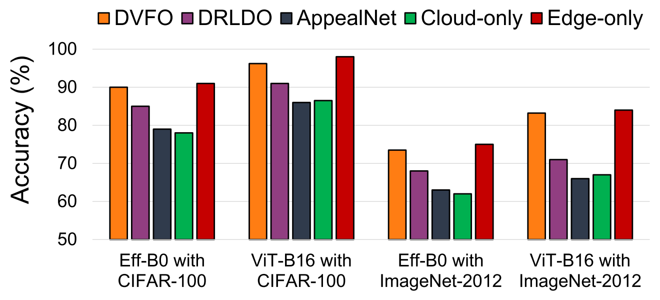

Fig. 11. Comparison of benchmark DNN inference accuracy on different datasets. We set the batch size to 1.

comparisons among different approaches. In addition, we use the same quantization (i.e., QAT) for AppealNet, DRLDO, and Cloud-only. All experimental results are reported on the edge devices listed in Table III. NVIDIA Xavier NX as the default edge device, unless otherwise mentioned. By default, we use $\eta = 0.5$ to balance energy consumption and end-to-end latency, and we test η from 0 to 1 in Section VI-E. The summation weight parameter λ is initialized to 0.5, and we also test λ from 0 to 1 in Section VI-E.

C. Comparison of Inference Performance

We first compare the inference performance of DVFO with baselines. We use trickle, a lightweight bandwidth control suite to set the transmission rate of the network bandwidth to 5 Mbps. Fig. 10 shows the performance comparison of EfficientNet-B0 and ViT-B16 DNN models on different datasets. We can see that DVFO consistently outperforms all baselines. To be more specific, the average energy consumption of these two DNN models using DVFO is 18.4%, 31.2%, 39.7%, and 43.4% lower than DRLDO, AppealNet, Cloud-only, and Edge-only, respectively. Meanwhile, DVFO significantly reduces the end-to-end latency by 28.6%$\sim$59.1% on average. Since the DNN is executed on edge devices, the end-to-end latency of Edge-only is higher than other approaches. Cloud-only is more sensitive to bandwidth fluctuations that leads to the highest end-to-end latency compared to other edge-cloud collaboration approaches.

Compared with DRLDO and AppealNet, the reduction of energy consumption and end-to-end latency mainly have the following two aspects: 1) *Co-optimization of frequency and proportion of offloading:* DRLDO only optimizes CPU frequency and offloading proportion. Since DVFO also takes GPU and memory frequencies as decision variables in DRL, which can optimize the hardware frequency to further reduce energy consumption and end-to-end latency. In addition, AppealNet does not utilize DVFS technology to optimize frequency, therefore its energy consumption is higher than DVFO. More importantly, binary offloading in AppealNet is more sensitive to network bandwidth than partial offloading. 2) *Lightweight offloading mechanism:* Compared with DRLDO and AppealNet that need to offload the original feature map to cloud servers, DVFO combines attention mechanism with quantization technology to compress the less important feature map to be offloaded, without losing much information.

Fig. 11 shows that DVFO can maintain similar inference accuracy to Edge-only (i.e., the loss of accuracy is within 2%), compared to other baseline methods with significant drop in accuracy. Note that Edge-only performs uncompressed original feature map and thus achieves the highest accuracy. Since AppealNet and Cloud-only leverage the same compression technique for binary offloading (i.e., compress the whole feature map), they suffer from similar accuracy loss, which significantly reduces accuracy. DRLDO leverages the partial offloading mechanism similar to DVFO. However, DVFO leverages a lightweight weighted summation-based fusion method, the accuracy, therefore, is higher than that of DRLDO. Such results illustrate the effective offloading for DVFO, which utilizes an attention mechanism to identify the less important features combined with high precision quantization aware training to minimize accuracy loss.

In Fig. 12, we report the trend of the hardware frequencies with inference process on EfficientNet-B0 and ViT-B16 under different datasets, respectively. We take the inference of EfficientNet-B0 in Fig. 12(a) under CIFAR-100 [8] dataset as an example for analysis. The whole end-to-end latency consists of ❶ edge inference, ❷ sum of offloading and compression, and ❸ cloud inference (including fusion operations). We can conclude from Fig. 2 that EfficientNet-B0 is memory-intensive DNN model, so that the frequencies of CPU and memory dominate edge inference, while the frequency of GPU has not yet come close to the performance bottleneck. In contrast, the ViT-B16 DNN model with compute-intensive properties has a significant increase in the GPU frequency during edge inference (Fig. 12(b) and (d)), which means that ViT-B16 can effectively utilize the GPU. Moreover, since DVFO adopts the attention-based lightweight compression mechanism in Section V-B and the concurrent offloading strategy with negligible overhead (i.e., *thinking-while-moving*) in Section V-A, the offloading and compression operations have extremely low hardware frequencies, which can save energy while reducing the offloading latency. For cloud inference, edge devices does not involve inference, offloading, and compression operations, so that DVFO only

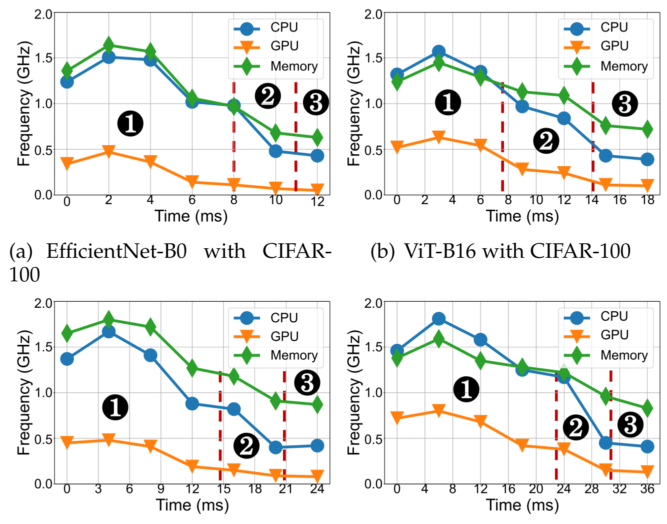
(a) EfficientNet-B0 with CIFAR-100 (b) ViT-B16 with CIFAR-100 (c) EfficientNet-B0 with ImageNet-2012 (d) ViT-B16 with ImageNet-2012

Fig. 12. Trend of hardware frequencies of NVIDIA Xavier NX with the execution process for EfficientNet-B0 and ViT-B16 under CIFAR-100 [8] and ImageNet-2012 [24] datasets, respectively. The execution process consists of ❶ edge inference, ❷ sum of offloading and compression, and ❸ cloud inference (including fusion operations).

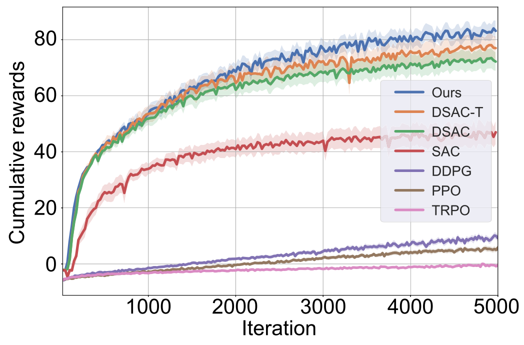
Fig. 13. Comparison of concurrent DQN in DVFO with five state-of-the-art (SOTA) and popular DRL algorithms.

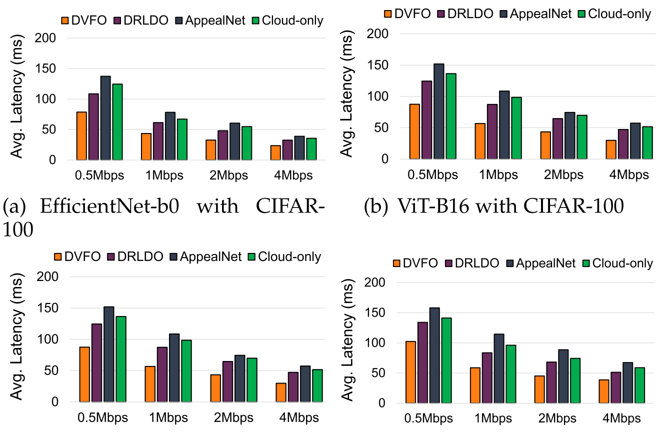
(a) EfficientNet-b0 with CIFAR-100 (b) ViT-B16 with CIFAR-100 (c) EfficientNet-b0 with ImageNet-2012 (d) ViT-B16 with ImageNet-2012

Fig. 14. End-to-end latency of EfficientNet-b0 and ViT-B16 for CIFAR-100 [21] and ImageNet-2012 [24] datasets under different network bandwidths.

needs to maintain the hardware frequencies at which the system normally operates.

Furthermore, we evaluate the policy performance (i.e., cumulative rewards) of our proposed concurrent DQN with *thinking-while-moving*, compared to six state-of-the-art (SOTA) and baseline DRL algorithms. As shown in Fig. 13, Distributional Soft Actor-Critic [41] (DSAC) introduces a value distribution learning principle to alleviate the overestimation problem [33] of the V-function and stabilize the learning process, based on the maximum entropy reinforcement learning framework, i.e., Soft Actor-Critic [42] (SAC). As a SOTA DRL algorithm, DSAC with three refinements (DSAC-T) [43] further inhibit the overestimation to improve the policy performance after convergence by leveraging expected value substituting, twin value distribution learning, and variance-based critic gradient adjusting. However, DSAC and DSAC-T are not applicable to a dynamic environment with sequential decisions. In contrast, our proposed concurrent DQN algorithm encourages the agent in DRL to explore more environmental states within the same time, enabling it to solve the co-optimization problem in DVFO more efficiently and consistently, its policy performance, therefore, outperforms DSAC and DSAC-T. In addition, compared to four baseline DRL algorithms, i.e., SAC [42] and DDPG [44], which are based on Off-policy (where data from historical policies can be utilized for learning), as well as PPO [45] and TRPO [46], which are based on On-policy (where data from the current policy can only be utilized for learning), our concurrent DQN has more significant policy performance improvements in dynamic concurrent environments.

D. Impact of Network Bandwidth

As mentioned before, edge-cloud network bandwidth may be a bottleneck for efficient feature map offloading, thus it is important to evaluate the performance of DVFO on different network bandwidth conditions. Due to energy and cost constraints, edge devices are equipped with *WiFi* modules that have lower transmission rates compared with cloud servers. Here we set the network bandwidth between 0.5 Mbps and 8 Mbps to simulate different network conditions.

The results in Fig. 14 illustrate that the trend of end-to-end latency for EfficientNet-b0 and ViT-B16 with various edge-cloud collaboration approaches using CIFAR-100 [21] and ImageNet-2012 [24] datasets under different network bandwidths. Benefit from offloading the less important feature map, the end-to-end latency of DVFO is lower than other baselines, even if the available network bandwidth is only 0.5 Mbps, which can effectively reduce the end-to-end latency by 27.3%∼44.6%. We also observe that the performance improvement of DVFO decreases when the network bandwidth increases. It means that the network bandwidth is no longer a bottleneck. The performance improvement is mainly the appropriate adjustment of the hardware frequency for edge devices.

In contrast, the performance of three baselines are highly dependent on network bandwidth. On the one hand, the binary offload mechanism of ApplealNet and Cloud-only offloads

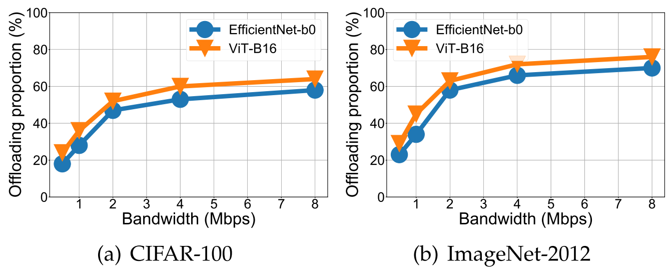

Fig. 15. Offloading proportion ξ of EfficientNet-b0 for CIFAR-100 [21] and ImageNet-2012 [24] datasets under different network bandwidths.

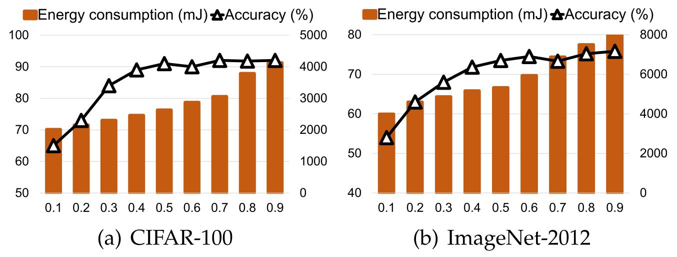

Fig. 16. Sensitivity analysis of the summation weight parameter λ on different datasets. We use EfficientNet-b0 [25] as a test case.

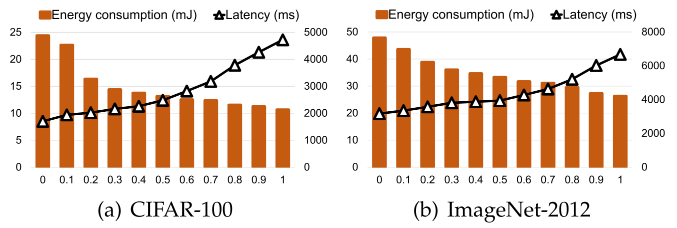

Fig. 17. Sensitivity analysis of weight parameter η on different datasets. We use EfficientNet-b0 [25] as a test case.

all data to the cloud servers. In particular, the hard-case discriminator of ApplealNet adds additional overhead compared to Cloud-only, which has the highest end-to-end latency, and Cloud-only takes second place. While DRLDO offloads part of the input data to cloud servers, the original data is not compressed. In addition, the think-while-moving concurrent offload mechanism in DVFO is faster than the conventional reinforcement learning-based offloading method in DRLDO, and thus has the lowest end-to-end latency. Overall, DVFO can make better adaptive adjustments to proportion of offloading and the hardware frequency of edge devices with the fluctuation of network bandwidth.

Furthermore, we report the trend of the offloading proportion parameter ξ on different bandwidth conditions. As shown in Fig. 15, the offloading proportion parameter ξ in DVFO increases with the bandwidth improvement. Specifically, only about 20% of the feature map are offloaded to the cloud servers when the bandwidth is 0.5 Mbps. This is due to the fact that costly communication latency at low bandwidth overlaps the benefits of offloading, so that DVFO tends to local inference. In constrast, the offloading proportion of DVFO is up to 70% at the 8 Mbps high bandwidth, which indicates that the offloading benefit is much higher than the communication overhead.

Note that the input size of different datasets and DNN models with different operation intensities also affect the offloading proportion of feature map. For instance, the offloading proportion for ImageNet-2012 dataset in Fig. 15(b) is 18.7% higher on average than the CIFAR-100 dataset in Fig. 15(a) with the same bandwidth. Similarly, the offloading proportion of ViT-B16 with two datasets is 12% higher on average than that of EfficientNet-b0 under different bandwidths. Since the input size in the ImageNet-2012 dataset and ViT-B16 are more complex than the CIFAR-100 dataset and EfficientNet-b0 respectively, it means that inference requires more computing resources. Therefore, DVFO tends to offload more feature map to cloud servers with abundant computing resources. In summary, DVFO is able to adaptively adjust the offloading proportion of feature map under different bandwidths, which also explains the performance improvement of DVFO with increasing bandwidth in Fig. 14.

E. Sensitivity Analysis

1) Impact of the Summation Weight Parameter λ: Taking EfficientNet-b0 [25] as an example, Fig. 16(a) and (b) shows the impact of the summation weight parameter λ on the performance of CIFAR-100 [21] and ImageNet-2012 [24] datasets, respectively. It can be seen that energy consumption and inference accuracy improve with the increase of λ. In particular, a smaller λ (≤ 0.2) significantly decreases accuracy, while a higher λ (≥ 0.8) sharply increases inference energy consumption. The intuition behind this is that a smaller λ reduces the contribution of important features locally, which misses some important information in inference and degrades accuracy. In contrast, increasing the value of λ, forces the majority of the inference tasks to the local DNN, which leads to higher energy consumption. Note that the optimal value of λ depends on the characteristics of the data in the training dataset. In practice, setting λ to an appropriate value between 0.4 and 0.6 can effectively reduce energy consumption while maintaining high accuracy.

2) Impact of the Relative Importance Coefficient η: In Fig. 17, we also take EfficientNet-b0 [25] as an example to show the impact of weight parameter η that trade-off between energy consumption and end-to-end latency, given different datasets. We observe that DVFO significantly reduces energy consumption with increasing values of η (≥ 0.1), while maintaining low end-to-end latency η (≤ 0.6). Specifically, compared to $\eta = 0.1$, even though DVFO reduced end-to-end latency by up to 39.2% at $\eta = 0.4$ on CIFAR-100 [8] dataset, the energy consumption is only increased by 16.5%. We also observe a similar phenomenon on the ImageNet-2012 [24] dataset. In summary, DVFO allows users to adjust the trade-off between energy consumption and end-to-end latency by selecting an appropriate weight parameter η.

F. Comparison of Various Fusion Methods

In this section, we compare the accuracy loss and runtime overhead (i.e., energy consumption and end-to-end latency) induced by weighted summation in DVFO and NN-based fusion methods, compared to single-device inference (without fusion).

TABLE IV
COMPARISON OF VARIOUS FUSION METHODS

Fusion methods	Accuracy (%)	
	CIFAR-100	ImageNet-2012
Single-device (without fusion)	91.84	74.52
Fully connected-based NN layer	87.39 (**4.45** ↓)	70.63 (**3.89** ↓)
Convolutional-based NN layer	82.93 (**8.91** ↓)	68.24 (**6.28** ↓)
DVFO (Ours)	**91.16 (0.68 ↓)**	**73.96 (0.56 ↓)**

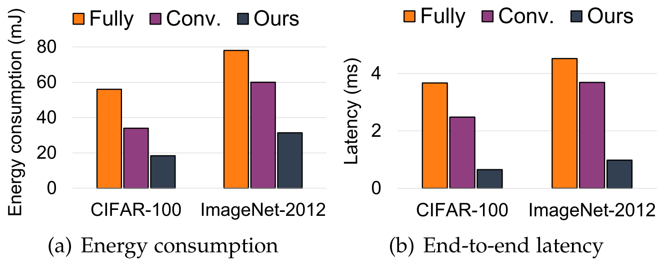

Fig. 18. Comparison of runtime overhead for different fuse methods.

In particular, it can be seen from Fig. 16 that the appropriate value of λ has different preferences under different datasets. To achieve high accuracy while maintaining low energy consumption, we set λ to 0.5 for CIFAR-100 [21] and 0.6 for ImageNet-2012 [24], respectively. We use a filter size of 3 × 3 and a softmax function to implement a convolutional layer and a fully connected layer for fusing inference results, respectively.

As shown in Table IV, the weighted summation we used in DVFO can achieve the lowest accuracy loss (within 1%), compared to single-device inference. This is due to weighted summation enables both the inference results of edge devices and cloud servers to maintain a highly consistent data alignment, and such lightweight point-to-point weighting has low-complexity with negligible overhead. The main challenge of weighted summation is that the output of Local DNN at edge devices and Remote DNN at cloud servers may potentially have a big difference. For instance, a few but the output values with high importance in Local DNN could be overlapped by the output values with less importance in Remote DNN. We can maintain λ in an appropriate range by manual fine-tuning or utilizing a learning-based adaptive strategy in DVFO, thereby minimizing the additional inference accuracy loss that may result. In contrast, neural network-based fusion approaches (i.e., fully connected layers and convolutional layers) have significant accuracy loss, which is 6.7 × and 12.3 × that of the weighted summation in DVFO, respectively. As pointed out in Section V-C, neural network-based fusion approaches break the alignment of weighted values and thus significantly reduce inference accuracy.

As shown the result in Fig. 18, we compare the runtime overhead of weighted summation and NN-based fusion methods (i.e., convolutional and fully connected layers). First, in terms of energy consumption in Fig. 18(a), compared to the NN-based fusion method, the energy consumption of weighted summation is reduced by 56.8% on average. Second, weighted summation reduces the average end-to-end latency by up to 77.5%, as shown in Fig. 18(b). It also illustrates the energy efficiency of

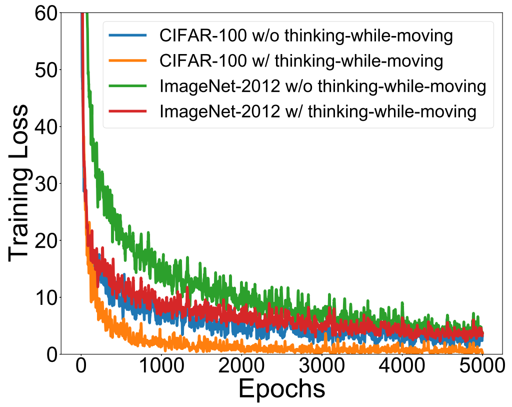

Fig. 19. Training performance of DVFO with/without a *thinking-while-moving* mechanism on CIFAR-100 [21] and ImageNet-2012 [24] datasets. We use EfficientNet-b0 [25] as a test case.

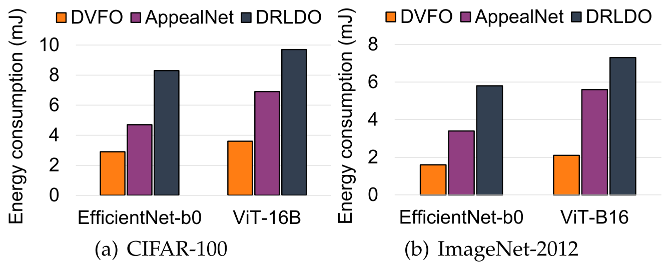

Fig. 20. Comparison of runtime overhead for different datasets.

such lightweight point-to-point fusion methods. In contrast, the NN-based fusion methods significantly reduce energy efficiency due to inherently expensive computation.

G. Overhead Analysis

1) Training Overhead: We first evaluate the training overhead comparison of DVFO with/without a *thinking-while-moving* training strategies, here we use EfficientNet-b0 [25] on CIFAR-100 [21] and ImageNet-2012 [24] datasets as a test case. As shown in Fig. 19, DVFO with *thinking-while-moving* shows faster convergence during the training procedure, indicating that although the attention module increases the complexity of learning, DVFO can still guarantee fast convergence of training by designing appropriate cost metrics and parallel strategies.

2) Runtime Overhead: The attention module in DVFO introduces additional runtime overhead. We evaluate the energy consumption of the attention module (i.e., SCAM) averaged over 10 inference. As shown in Fig. 20, DVFO consumes less energy due to uses an extremely lightweight attention module. The energy consumption of DVFO is 38%∼62% lower than AppealNet and 63%∼71% lower than DRLDO.

H. Evaluation of Scalability

In this section, we evaluate the scalability of DVFO for various DNN models. Note that we performance extensive experiments on heterogeneous edge devices (i.e., Jetson Nano,

TABLE V
Evaluation of Scalability for Edge Inference Services

Edge Device	Model	End-to-end latency (ms)			Energy consumption (mJ)			Accuracy loss (%)		
		AppealNet	DRLDO	**DVFO**	AppealNet	DRLDO	**DVFO**	AppealNet	DRLDO	**DVFO**
NVIDIA Jetson Nano	ResNet-18	24.1	20.6	**14.8**	6217	5723	**4867**	4.62	2.13	**0.54**
	Inception-v4	25.6	24.9	**16.7**	6894	6421	**5346**	5.14	3.47	**0.98**
	MobileNet-v2	30.7	28.4	**17.6**	7082	6804	**5623**	2.84	1.69	**0.63**
	YOLOv3-Tiny	26.8	24.7	**18.4**	6489	6174	**5391**	3.87	2.26	**0.68**
	RetinaNet	35.4	31.6	**25.9**	8256	7498	**6157**	2.65	1.54	**0.78**
	DeepSpeech	16.6	14.3	**12.5**	5733	5289	**4826**	2.24	1.28	**0.35**
	Average	**26.5 (+49.7%)**	**24.1 (+36.2%)**	17.7	**6779 (+53.0%)**	**6318 (+42.6%)**	4431	**3.56 (5.4×)**	**2.06 (3.12×)**	0.66
NVIDIA Jetson TX2	ResNet-18	18.9	14.6	**10.8**	6897	6438	**5278**	3.14	1.87	**0.48**
	Inception-v4	16.3	14.9	**12.1**	7018	6381	**5469**	2.36	1.65	**0.49**
	MobileNet-v2	21.6	18.4	**14.9**	8248	7456	**6597**	3.71	2.59	**1.35**
	YOLOv3-Tiny	19.4	15.1	**12.7**	6732	6279	**5367**	1.95	1.46	**0.82**
	RetinaNet	27.6	21.5	**16.2**	9546	8948	**7294**	3.36	2.08	**1.23**
	DeepSpeech	12.4	10.7	**8.3**	6754	6017	**5309**	2.46	1.61	**0.27**
	Average	**19.4 (+55.2%)**	**15.9 (+27.2%)**	12.5	**7533 (+30.0%)**	**6920 (+17.6%)**	5886	**2.83 (3.68×)**	**1.88 (2.44×)**	0.77
NVIDIA Jetson Orin NX	ResNet-18	6.4	5.9	**3.6**	13421	12904	**10543**	2.68	1.45	**0.37**
	Inception-v4	5.9	5.3	**3.2**	13862	12976	**10863**	1.97	1.34	**0.41**
	MobileNet-v2	7.2	6.9	**4.1**	16235	15786	**12973**	2.98	2.37	**0.86**
	YOLOv3-Tiny	6.9	6.2	**3.7**	14732	13698	**11765**	1.59	1.24	**0.67**
	RetinaNet	7.8	7.1	**5.6**	17032	16467	**12951**	2.67	1.85	**0.92**
	DeepSpeech	5.7	4.9	**3.1**	12547	11638	**10236**	2.08	1.37	**0.21**
	Average	**6.7 (+41.8%)**	**6.1 (+36.0%)**	3.9	**14578 (+20.7%)**	**13912 (+16.9%)**	11555	**2.33 (4.09×)**	**1.60 (2.81×)**	0.57
NVIDIA Jetson AGX Orin	ResNet-18	3.6	3.1	**2.1**	23246	21529	**16438**	2.13	1.17	**0.24**
	Inception-v4	3.4	3.3	**1.9**	21358	20469	**15962**	1.68	1.54	**0.27**
	MobileNet-v2	4.2	3.9	**2.6**	26124	25394	**17341**	2.38	1.46	**0.52**
	YOLOv3-Tiny	3.8	3.6	**2.3**	24258	22964	**16057**	1.26	0.94	**0.41**
	RetinaNet	4.5	4.2	**2.9**	29571	27837	**18725**	2.14	1.48	**0.73**
	DeepSpeech	3.1	2.8	**1.8**	20469	18967	**15143**	1.76	1.07	**0.15**
	Average	**3.8 (+39.4%)**	**3.5 (+34.3%)**	2.3	**24171 (+31.3%)**	**22860 (+27.4%)**	16611	**1.89 (4.85×)**	**1.28 (3.28×)**	0.39

Jetson TX2, Orin NX and AGX Orin in Table III), but our DVFO framework is also applicable to homogeneous edge devices in general. We take the widely-deployed deep learning models, ResNet-18 [34], Inception-v3 [47], and MobileNet-v2 [48] as the image classification services, the widely-deployed deep learning models, YOLOv3-Tiny [49] and RetinaNet [50], as the object detection services, and the widely-deployed deep learning model, DeepSpeech [51], as the speech recognition service. The evaluation results are reported in Table V. We can conclude that DVFO consistently outperforms AppealNet and DRLDO in terms of end-to-end latency, energy consumption, and accuracy loss, respectively. To be more specific, DVFO reduces the average end-to-end latency by 49.7% and 36.2% on Jetson Nano, compared with AppealNet and DRLDO, respectively, and the performance improvement remains consistent across the other three edge devices with high TOPS metric (39.4%$\sim$55.2% and 27.2%$\sim$36.2% respectively). For energy consumption, compared to AppealNet and DRLDO, DVFO achieves energy-saving of up to 42.6% and 53% for Jetson Nano on average, respectively. Since the energy efficiency of other edge devices (i.e., TX2, Orin NX, and AGX Orin) is lower than that of Jetson Nano, the energy-saving is conservative, but still better than the baseline (16.9%$\sim$31.3%). As mentioned in Section VI-F, benefit from the efficient fusion method based on weighted summation, the average accuracy loss of DVFO on different datasets and heterogeneous edge devices remains within 1%, which is much lower than that of AppealNet and DRLDO ($2.44\times \sim 5.4\times$). Overall, DVFO can seamlessly adapt to heterogeneous edge devices and various widely-deployed DNN models, and thus it has flexible scalability.

VII. Related Work

A. Learning-Based DVFS

Prior work [13], [15], [17], [18], [20], [52], [53], [54], [55] has proposed a series of deep reinforcement learning-based DVFS techniques to reduce energy consumption. For instance, DRL quality optimizer [13] combines deep reinforcement learning-based DVFS technology with LSTM-based selectors to reduce end-to-end latency and improve quality of service (QoS). QL-HDS [18] combines Q-learning with stacked auto-encoder, and proposes a hybrid DVFS energy-saving scheduling scheme based on Q-learning. DQL-EES [53] and Double-Q governor [54] leverage double-Q learning-based DVFS technology that dynamically scale computing frequency to achieve efficient energy-saving. Hybrid DVFS [52] considers heterogeneous workloads, dynamic relaxation and power constraints, which utilizes reinforcement learning-based hybrid DVFS technology to achieve energy-saving. CARTAD [17] leverages reinforcement learning-based task scheduling and DVFS to jointly optimize end-to-end latency and temperature on multi-core CPUs systems. Ring-DVFS [55] proposes an enhanced reinforcement learning-based DVFS technique to reduce power consumption on multi-core CPU systems.

Most related to our work is DRLDO [15], a data offloading scheme that combines DRL and DVFS to reduce the energy consumption of IoT devices. However, the above-mentioned DRL-based DVFS approaches including [15] only optimize the CPU frequency of edge devices, ignoring the impact of GPU and memory frequencies on energy consumption. Moreover, DRLDO [15] offloads uncompressed raw data to cloud servers, causing system instability and bandwidth bottlenecks. In this work, we introduce DVFS into edge-cloud collaborative architecture, i.e., using DVFS for DNN feature map offloading to further improve the energy-saving effect of edge devices. In addition, we utilize the attention mechanism to efficiently compress the original DNN feature map, reducing the transmission delay of compressed DNN feature map to be offloaded while maintaining accuracy.

B. *Edge-Cloud Collaborative DNN Model Inference*

Since edge devices are usually resource-constrained, it is necessary to utilize cloud servers with abundant computing resources for edge-cloud collaborative inference to reduce end-to-end latency. Existing studies [11], [56] have been revealed that transmission of DNN feature map is a major network bottleneck for offloading. Therefore, prior work [11], [12], [14], [56] has proposed various collaborative inference methods that combine a series of compression techniques to reduce the transmission of DNN feature map. For instance, DeepCOD [14] and Starfish [56] designs efficient encoders and decoders based on compressed sensing theory and application-specific codecs, respectively, and then offloads the compressed data from the local to the edge server, thereby effectively reducing end-to-end delays. AgileNN [12] uses attention mechanism to identify the importance of DNN feature map, and it reduces the end-to-end latency by offloading a large number of compressed less important features to remote. SPINN [11] achieves the progressive inference for edge-cloud collaboration by placing multiple early-exit points in the neural network, which not only considers the resource-constrained local devices, but also takes into account the instability and communication costs of the cloud.

In addition, [57] proposes a pipelined scheme for collaborative inference on a heterogeneous IoT edge cluster to reduce redundant calculation and communication overhead in order to maximize the throughput. DCCI [58] and AppealNet [19] perform binary offloading (local inference only or full offloading to the cloud servers) on the input data based on the hard-case discriminator. ELF [59] splits a single video frame and offloads the segmented local video frames to multiple edge servers, which accelerates parallel inference for high resolution vision models. CNNPC [60] jointly optimizes model partitioning and compression, which significantly speeds up collaborative inference with the end-edge-cloud computing paradigm. However, these approaches do not incorporate optimization of hardware frequency for better energy saving. Our approach combines the advantages of DVFS and feature map offloading.

C. *On-Device DNN Model Inference*

The MLPerf Mobile Inference Benchmark [61] reveals impressive progress in on-device inference [62], benefiting from the synergistic impact of increasing performance at the edge, highly flexible lightweight models, and efficient deep learning frameworks. We highlight that previous work [62], [63], [64], [65], [66], [67], [68], [69] can also achieve effective energy saving and low end-to-end latency only through efficient on-device inference, except for edge-cloud collaborative inference. Mistify [63] and NeuralUCB [64] studied the automated customization for on-device DL inference, which reduce DNN manual porting time and improve quality of experience (QoE) by automatically porting cloud-based compressed models to edge devices and online learning algorithms, respectively. MEmCom [65] significantly improves the on-device inference performance of recommendation models by using a model compression technique based on multi-embedding compression. DeiT-Tiny [66] is the first empirical study on efficient on-device inference for visual transformer, reducing the end-to-end latency by removing redundant attention heads and forward neural network layers. eNODE [67] achieves efficient on-device inference for neural differential equations (NODEs) by architecture-algorithm co-design. BlastNet [68] leverages dual-block based fine-grained dynamic scheduling to enable on-device real-time multi-model inference across CPU-GPU. Similarly, POS [69] leverages operator granularity-oriented computational graph optimization with reinforcement learning to accelerate multi-model real-time on-device inference. AsyMo [62] leverages model partitioning based on cost-model and asymmetric task scheduling for mobile CPUs to enable energy-efficient on-device inference.

As privacy security for on-device inference becomes increasingly challenging, ShadowNet [70] leverages a trusted execution environment (TEE) to preserve model privacy while ensuring efficient inference. Note that on-device inference is orthogonal to our work, which can further reduce end-to-end latency and energy consumption.

VIII. Conclusion

In this work, we propose DVFO, an DVFS enabled learning-based energy-efficient collaborative inference framework. DVFO co-optimizes the CPU, GPU, and memory frequencies of edge devices, as well as the proportion of feature map to be offloaded to cloud servers. We apply concurrent control mechanism named *thinking-while-moving* in learning-based optimization, and propose an importance-based feature map offloading scheme by leveraging a *spatial-channel attention* mechanism, to accelerate convergence and alleviate edge-cloud network bottlenecks, respectively. Extensive evaluations on widely-deployed DNN models with three domain-specific on five heterogeneous edge devices show that DVFO significantly outperforms existing offloading schemes in terms of energy consumption and end-to-end latency, while maintaining high accuracy.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[2] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[3] N. Zeng, P. Wu, Z. Wang, H. Li, W. Liu, and X. Liu, "A small-sized object detection oriented multi-scale feature fusion approach with application to defect detection," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–14, 2022.

[4] M. Kim, A. K. Jain, and X. Liu, "AdaFace: Quality adaptive margin for face recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 18750–18759.

[5] Z. Liu, G. Lan, J. Stojkovic, Y. Zhang, C. Joe-Wong, and M. Gorlatova, “CollabAR: Edge-assisted collaborative image recognition for mobile augmented reality,” in *Proc. IEEE/ACM 19th Int. Conf. Inf. Process. Sensor Netw.*, 2020, pp. 301–312.

[6] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, “R-TOD: Real-time object detector with minimized end-to-end delay for autonomous driving,” in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 191–204.

[7] M. Han, H. Zhang, R. Chen, and H. Chen, “Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences,” in *Proc. 16th USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 539–558.

[8] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” University of Toronto, Toronto, Ontario, Canada, Tech. Rep., 2009.

[9] Q. Wang, X. Mei, H. Liu, Y.-W. Leung, Z. Li, and X. Chu, “Energy-aware non-preemptive task scheduling with deadline constraint in DVFS-enabled heterogeneous clusters,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4083–4099, Dec. 2022.

[10] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, “Coordinated batching and DVFS for DNN inference on GPU accelerators,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2496–2508, Oct. 2022.

[11] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, “SPINN: Synergistic progressive inference of neural networks over device and cloud,” in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–15.

[12] K. Huang and W. Gao, “Real-time neural network inference on extremely weak devices: Agile offloading with explainable AI,” in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, 2022, pp. 200–213.

[13] F. Chen, H. Yu, W. Jiang, and Y. Ha, “Quality optimization of adaptive applications via deep reinforcement learning in energy harvesting edge devices,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4873–4886, Nov. 2022.

[14] S. Yao et al., “Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency,” in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 476–488.

[15] S. K. Panda, M. Lin, and T. Zhou, “Energy efficient computation offloading with DVFS using deep reinforcement learning for time-critical IoT applications in edge computing,” *IEEE Internet Things J.*, vol. 10, no. 8, pp. 6611–6621, Apr. 2023.

[16] J. Dodge et al., “Measuring the carbon intensity of AI in cloud instances,” in *Proc. ACM Conf. Fairness Accountability Transparency*, 2022, pp. 1877–1894.

[17] D. Liu, S.-G. Yang, Z. He, M. Zhao, and W. Liu, “CARTAD: Compiler-assisted reinforcement learning for thermal-aware task scheduling and DVFS on multicores,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1813–1826, Jun. 2022.

[18] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, “Energy-efficient scheduling for real-time systems based on deep q-learning model,” *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 132–141, First Quarter 2019.

[19] M. Li, Y. Li, Y. Tian, L. Jiang, and Q. Xu, “AppealNet: An efficient and highly-accurate edge/cloud collaborative architecture for DNN inference,” in *Proc. IEEE/ACM 58th Des. Automat. Conf.*, 2021, pp. 409–414.

[20] C. Lin, K. Wang, Z. Li, and Y. Pu, “A workload-aware DVFS robust to concurrent tasks for mobile devices,” in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–16.

[21] O. M. Andrychowicz et al., “Learning dexterous in-hand manipulation,” *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[22] T. Xiao et al., “Thinking while moving: Deep reinforcement learning with concurrent control,” in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[23] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional block attention module,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–19.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[25] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 6105–6114.

[26] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” 2020, *arXiv:2010.11929*.

[27] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[28] J. You, J.-W. Chung, and M. Chowdhury, “Zeus: Understanding and optimizing GPU energy consumption of DNN training,” in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation*, 2023, pp. 119–139.

[29] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, “Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 360–374, Jan. 2021.

[30] X. Li et al., “Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference,” in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 8657–8665.

[31] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, “zTT: Learning-based DVFS with zero thermal throttling for mobile devices,” in *Proc. 19th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2021, pp. 41–53.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[33] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[35] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2921–2929.

[36] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 618–626.

[37] Y. Song, T. Wang, P. Cai, S. K. Mondal, and J. P. Sahoo, “A comprehensive survey of few-shot learning: Evolution, applications, challenges, and opportunities,” *ACM Comput. Surv.*, vol. 55, 2023, Art. no. 271.

[38] F. Zhuang et al., “A comprehensive survey on transfer learning,” *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.

[39] L. Bottou, “Stochastic gradient descent tricks,” in *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Germany: Springer, 2012, pp. 421–436.

[40] jetson-stats, Accessed: Nov. 26, 2023. [Online]. Available: https://github.com/rbonghi/jetson_stats

[41] J. Duan, Y. Guan, S. E. Li, Y. Ren, Q. Sun, and B. Cheng, “Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6584–6598, Nov. 2022.

[42] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 1861–1870.

[43] J. Duan, W. Wang, L. Xiao, J. Gao, and S. E. Li, “DSAC-T: Distributional soft actor-critic with three refinements,” 2023, *arXiv:2310.05858*.

[44] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” 2015, *arXiv:1509.02971*.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017, *arXiv:1707.06347*.

[46] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2015, pp. 1889–1897.

[47] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[48] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[49] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[50] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.

[51] A. Hannun et al., “Deep speech: Scaling up end-to-end speech recognition,” 2014, *arXiv:1412.5567*.

[52] F. M. M. ul Islam and M. Lin, “Hybrid DVFS scheduling for real-time systems based on reinforcement learning,” *IEEE Syst. J.*, vol. 11, no. 2, pp. 931–940, Jun. 2017.

[53] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, “A double deep q-learning model for energy-efficient edge scheduling,” *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 739–749, Sep./Oct. 2019.

[54] H. Huang, M. Lin, L. T. Yang, and Q. Zhang, “Autonomous power management with double-q reinforcement learning method,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1938–1946, Mar. 2020.

[55] A. Yeganeh-Khaksar, M. Ansari, S. Safari, S. Yari-Karin, and A. Ejlali, "Ring-DVFS: Reliability-aware reinforcement learning-based DVFS for real-time embedded systems," *IEEE Embedded Syst. Lett.*, vol. 13, no. 3, pp. 146–149, Sep. 2021.

[56] P. Hu, J. Im, Z. Asgar, and S. Katti, "Starfish: Resilient image compression for AIoT cameras," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 395–408.

[57] X. Yang, Q. Qi, J. Wang, S. Guo, and J. Liao, "Towards efficient inference: Adaptively cooperate in heterogeneous IoT edge cluster," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 12–23.

[58] Y. Hu, Z. Li, Y. Chen, Y. Cheng, Z. Cao, and J. Liu, "Content-aware adaptive device-cloud collaborative inference for object detection," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 19087–19101, Nov. 2023.

[59] W. Zhang et al., "ELF: Accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 201–214.

[60] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "CNNPC: End-edge-cloud collaborative CNN inference with joint model partition and compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4039–4056, Dec. 2022.

[61] V. Janapa Reddi et al., "MLPerf mobile inference benchmark: An industry-standard open-source machine learning benchmark for on-device AI," *Proc. Mach. Learn. Syst.*, vol. 4, pp. 352–369, 2022.

[62] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "AsyMo: Scalable and efficient deep-learning inference on asymmetric mobile CPUs," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 215–228.

[63] P. Guo, B. Hu, and W. Hu, "Mistify: Automating DNN model porting for on-device inference at the edge," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 705–719.

[64] Y. Bai, L. Chen, S. Ren, and J. Xu, "Automated customization of on-device inference for quality-of-experience enhancement," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1329–1342, May 2023.

[65] N. Pansare et al., "Learning compressed embeddings for on-device inference," in *Proc. Mach. Learn. Syst.*, vol. 4, pp. 382–397, 2022.

[66] X. Wang, L. L. Zhang, Y. Wang, and M. Yang, "Towards efficient vision transformer inference: A first study of transformers on mobile devices," in *Proc. 23rd Annu. Int. Workshop Mobile Comput. Syst. Appl.*, 2022, pp. 1–7.

[67] J. Zhu, Y. Tao, and Z. Zhang, "eNODE: Energy-efficient and low-latency edge inference and training of neural odes," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2023, pp. 802–813.

[68] N. Ling, X. Huang, Z. Zhao, N. Guan, Z. Yan, and G. Xing, "BlastNet: Exploiting duo-blocks for cross-processor real-time DNN inference," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, 2022, pp. 91–105.

[69] Z. Zhang, H. Li, Y. Zhao, C. Lin, and J. Liu, "POS: An operator scheduling framework for multi-model inference on edge intelligent computing," in *Proc. 22nd Int. Conf. Inf. Process. Sensor Netw.*, 2023, pp. 40–52.

[70] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 1489–1505.

Ziyang Zhang (Student Member, IEEE) received the MS degree from the School of Electronic Information and Optical Engineering, Nankai University, Tianjin, China, in 2020. He is currently working toward the PhD degree with the School of Computer Science and Technology, Harbin Institute of Technology (HIT), Harbin, China. His research interests include edge computing, machine learning system, and deep learning.

Yang Zhao (Senior Member, IEEE) received the BS degree in electrical engineering from Shandong University, in 2003, the MS degree in electrical engineering from the Beijing University of Aeronautics and Astronautics, in 2006, and the PhD degree in electrical and computer engineering from the University of Utah, in 2012. He was a lead research engineer with GE Global Research between 2013 and 2021. Since 2021, he has been with the Harbin Institute of Technology, Shenzhen, where he is a research professor with the International Research Institute for Artificial Intelligence. His research interests include wireless sensing, edge computing, and cyber physical systems.

Huan Li (Senior Member, IEEE) received the PhD degree in computer science from the University of Massachusetts at Amherst, in 2006. Her current research interests include AIoT, edge intelligence, distributed real-time systems, and data science. She has served as program committee member for numerous international conferences including IEEE RTAS, ICDCS, RTCSA, etc.

Changyao Lin (Student Member, IEEE) received the BS and MS degrees from the School of Computer Science and Technology, Harbin Institute of Technology (HIT), Harbin, China, in 2020 and 2022, respectively. He is currently working toward the PhD degree with HIT. His research interests include edge computing, distributed system, and deep learning.

Jie Liu (Fellow, IEEE) is a chair professor with the Harbin Institute of Technology Shenzhen (HIT Shenzhen), China and the dean of its AI Research Institute. Before joining HIT, he spent 18 years with Xerox PARC and Microsoft. He was a principal research manager with Microsoft Research, Redmond and a partner of the company. His research interests are cyber-physical systems, AI for IoT, and energy-efficient computing. He received IEEE TCCPS Distinguished Leadership Award and 7 Best Paper Awards from top conferences. He is an ACM Distinguished Scientist, and founding chair of ACM SIGBED China.