Ace-Sniper: Cloud–Edge Collaborative Scheduling Framework With DNN Inference Latency Modeling on Heterogeneous Devices

Weihong Liu, Jiawei Geng, Zongwei Zhu, Yang Zhao, Cheng Ji, Changlong Li, Zirui Lian, and Xuehai Zhou, *Member, IEEE*

***Abstract*—The cloud–edge collaborative inference requires efficient scheduling of artificial intelligence (AI) tasks to the appropriate edge intelligence devices. Deep neural network (DNN) inference latency has become a vital basis for improving scheduling efficiency. However, edge devices exhibit highly heterogeneous due to the differences in hardware architectures, computing power, etc. Meanwhile, the diverse DNNs are continuing to iterate over time. The diversity of devices and DNNs introduces high computational costs for measurement methods, while invasive prediction methods face significant development efforts and application limitations. In this article, we propose and develop Ace-Sniper, a scheduling framework with DNN inference latency modeling on heterogeneous devices. First, to address the device heterogeneity, a unified hardware resource modeling (HRM) is designed by considering the platforms as black-box functions that output feature vectors. Second, neural network similarity (NNS) is introduced for feature extraction of diverse and frequently iterated DNNs. Finally, with the results of HRM and NNS as input, the performance characterization network is designed to predict the latencies of the given unseen DNNs on heterogeneous devices, which can be combined into most time-based scheduling algorithms. Experimental results show that the average relative error of DNN inference latency prediction is 11.11%, and the prediction accuracy reaches 93.2%. Compared with the nontime-aware scheduling methods, the average waiting time for tasks is reduced by 82.95%, and the platform throughput is improved by 63% on average.**

***Index Terms*—Cloud–edge collaborative, hardware resource modeling (HRM), heterogeneous platform, inference latency modeling.**

Manuscript received 27 October 2022; revised 21 March 2023 and 23 June 2023; accepted 6 September 2023. Date of publication 12 September 2023; date of current version 22 January 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62102390 and Grant 62102179; in part by the National Key Laboratory of Science and Technology on Space Microwave under Grant HTKJ2022KL504021; in part by the Special Fund for Jiangsu Natural Resources Development (Innovation Project of Marine Science and Technology) under Grant JSZRHYKJ202218; and in part by the Natural Science Foundation of Jiangsu Province under Grant BK20200462. This article was recommended by Associate Editor M. Zapater. *(Corresponding author: Zongwei Zhu.)*

Weihong Liu, Jiawei Geng, Zirui Lian, and Xuehai Zhou are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: lwh2017@mail.ustc.edu.cn; gjw1998@mail.ustc.edu.cn; lzrustc@mail.ustc.edu.cn; xhzhou@ustc.edu.cn).

Zongwei Zhu is with the School of Software Engineering, University of Science and Technology of China, Hefei 230026, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: zzw1988@ustc.edu.cn).

Yang Zhao is with the National Key Laboratory of Science and Technology on Space Micrwave, Xi'an Institute of Space Radio Technology, Xi'an 710199, China (e-mail: yangzhao67@cast504.com).

Cheng Ji is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: cheng.ji@njust.edu.cn).

Changlong Li is with the School of Computer Science and Technology, East China Normal University, Shanghai 200063, China (e-mail: clli@cs.ecnu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2023.3314388

I. Introduction

THE CLOUD–EDGE collaborative architecture [1] (e.g., sedna [2] and kubeedge [3]) provides edge intelligence services close to the data source and works with cloud servers to provide intelligent computing. As shown in Fig. 1, the data generated at each edge domain is commonly private and cannot be uploaded to the cloud. Meanwhile, expensive communication costs [4] and increasing data volumes [5] make it unrealistic to provide inference services to intradomain clients from the distant cloud. Therefore, it is necessary to schedule and deploy deep neural networks (DNNs) from the cloud to the edge devices within the corresponding domain [6].

In recent years, to improve the efficiency of cloud–edge collaborative inference, many scheduling methods have been proposed in [7], [8], [9], [10], [11], and [12]. Most current research focuses on the scheduling methods based on time awareness [7], [8], [9], [10] owing to their better performance in terms of throughput and quality of service (QoS) than those without time awareness [11], [12]. However, as shown in Fig. 1, there usually exists a wide range of heterogeneous devices within each domain that differ in hardware architectures, floating-point operations per second (FLOPS), inference frameworks, software versions, and other factors [13], [14], [15]. These variations have a significant impact on the inference latency of different DNNs [16]. The heterogeneity of devices and the diversity of DNNs generate a huge search space for cloud–edge collaborative scheduling, leading to square-level complexity in scheduling [8], [9], [10], which also poses a huge challenge in obtaining the inference latency of diverse DNNs.

The current scheduling work mainly obtains DNN inferred latencies from two methods: 1) actual measurement and 2) prediction. Actual measurement methods [7], [8], [9], [10] involve deploying DNNs on each device to obtain accurate inference latency, but these methods become impractical as

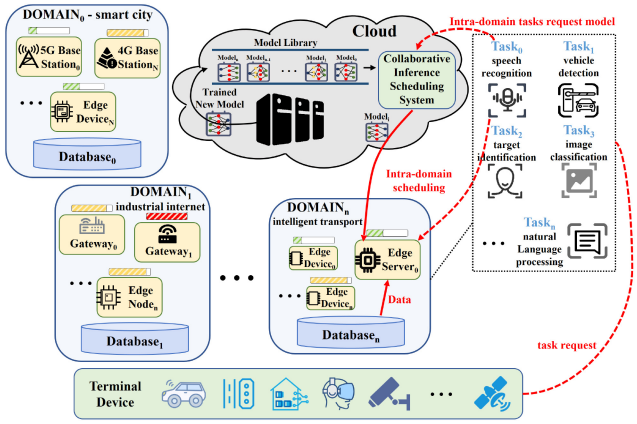

Fig. 1. Illustration of cloud–edge collaborative inference.

the measurement overhead continues to increase exponentially with the growing number of devices and DNNs. On the other hand, numerous works focus on the invasive prediction of DNN inference latencies. For example, some works [16], [17] analyze the latency of each operator and its combination on different devices to achieve highly accurate inference latency predictions. However, for most cloud–edge collaborative inference scenarios, it is not feasible to require open-source code for inference framework on each heterogeneous device, which makes it impossible to perform layer-by-layer analysis of DNNs. Moreover, the significant impact of platform heterogeneity on DNN inference latency makes it impractical to predict inference latency using actual measurements or invasive predictions.

To reduce the computational cost imposed by invasive prediction methods, noninvasive methods have been proposed in other works [18], [19], [20], [21]. For example, the FLOPS-based method has been widely applied to evaluate the efficiency [18], [19], [20], which is simple but not a direct metric of latency. The method in [21] successfully predicts inference latency by a noninvasive method. However, they consider simply the basic hardware parameters to characterize different devices, failing to work between different hardware architectures (e.g., GPUs and NPUs). Even both on GPUs, different inference frameworks and software versions can also cause poor prediction accuracy. In [22], a latency predictor (HELP) is proposed for heterogeneous devices, which directly uses a small number of DNN latency measurements as hardware embedding. However, HELP performs poorly in latency prediction due to the lack of considering the complex inference performance of DNNs on highly heterogeneous platforms. The above methods cannot guarantee prediction accuracy due to the lack of a reasonable hardware resource modeling (HRM) method.

It is discerned that similar DNN structural features can lead to similar DNN performance in a single device [21]. Inspired by this, we find that DNN inference latency can be predicted theoretically by performance characterization methods, thus narrowing the search space of the scheduling algorithm. Therefore, this article proposes *Ace-Sniper*, a scheduling framework with DNN inference latency modeling on heterogeneous devices, which accurately predicts DNN latencies on heterogeneous devices and significantly improves the efficiency of cloud–edge collaborative scheduling.

The results show that the average relative error of DNN inference latency prediction is 11.11%, and the prediction accuracy reaches 93.2%, which is 21.1% higher than that of Sniper. Compared with the methods without time awareness, the waiting time of tasks is reduced by 82.95% on average, and the clusters throughput is improved by 63% on average. More specifically, our main contributions are as follows.

1) This article proposes an HRM to achieve uniform modeling of highly heterogeneous platforms by analyzing the platform as a black box, which outputs a feature vector that effectively characterizes the inference computing power of platforms.
2) With the modeling results of HRM and neural network similarity (NNS) as input, the noninvasive performance characterization network (PCN) is improved to predict DNN inference latency on highly heterogeneous platforms without excessive computational costs and development works.
3) To fully represent platform heterogeneity in HRM, we introduce a new dimension (inference data flow) to the network structure analyzer and enrich the parameterized model of the random network generator.
4) By combining PCN and scheduling algorithm, a scheduling framework with DNN inference latency modeling (*Ace-Sniper*) is proposed and evaluated on GPUs and NPUs to demonstrate the superior latency prediction accuracy, enhanced throughput, and improved QoS.

II. Motivation

This section discusses the three observations of DNN inference scheduling techniques, which motivates the design of PCN and *Ace-Sniper*.

A. Time Awareness in Scheduling

For such NP-hard problems as scheduling multiple inference tasks on different devices, we perform experiments on ten computing nodes consisting of five types of different computing platforms (NVIDIA AGX, TX2, NX, Cambricon MLU220, and MLU270) for nine inference tasks. To explore the scheduling efficiency of different algorithms in heterogeneous systems, the nontime-aware algorithms (*Polling* [12] and *EDF* [11]) and the time-aware algorithm (*SLO-Scheduler* [10]) are included in the experiment. This experiment specifies two metrics for evaluating the algorithm performance: 1) transactions per second (TPS) and 2) standard deviation of nodes load [23]. The TPS can be used as a quantitative indicator of the throughput, and the *standard deviation for load balancing* measures the load balancing effect of the edge cluster.

As shown in Fig. 2(a), as the number of tasks per second (T) increases, the *EDF* and *Polling* algorithms, which lack time awareness, experience load imbalance and increase rapidly as the number of tasks per second (T) increases. However, *SLO-Scheduler* based on obtaining inference time with real measurements significantly outperforms the *EDF* and

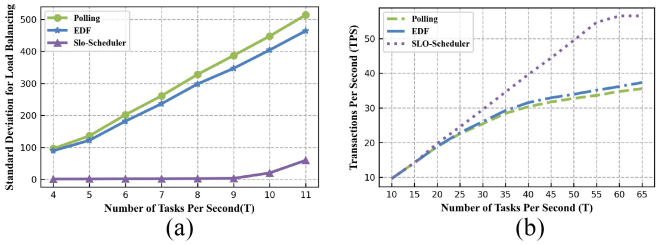

Fig. 2. Scheduling performance at different T. (a) Standard deviations of nodes load. (b) Tps of the edge cluster.

Polling algorithms in load balancing. In addition, as shown in Fig. 2(b), when $T = 65$, the TPS of *EDF* (37.4) and *Poling* (35.6) is also lower than that of *SLO-Scheduler* (56.3). Consequently, the first observation can be drawn as follows.

Observation 1: The inference time is necessary for improving scheduling efficiency. Nontime-aware algorithms can result in a load imbalance in highly heterogeneous edge cluster. This imbalance leads to underutilization of the computing power of the cluster and subsequently results in low throughput (TPS). To improve scheduling performance, it is necessary to obtain the inference latency of DNNs in advance.

B. Overhead of Time Measurement

According to works [24], [25], [26], deep learning algorithms have become widely used in various practical applications, leading to the emergence of numerous DNN models with different network structures. Additionally, different hardware architectures are being integrated into edge devices to meet diverse requirements in terms of performance and energy efficiency [13], [14]. Estimates indicate that over 100 companies are focusing on the development of specialized inference hardware architectures [15]. The problem of deploying rapidly iterating DNNs on continuously updated heterogeneous platforms for accurate inference latency prediction has become a major challenge. As a result, many scheduling approaches [7], [10], [27] fall short in fully leveraging the computing power of heterogeneous platforms when scheduling multiple inference tasks, due to intolerable waiting delays and the absence of reliable inference latency prediction methods.

Observation 2: The actual deployment to get the inference time brings huge overhead. It is not feasible to obtain the latency by measurement since DNNs and devices are continuously updating and iterating. Therefore, there is a need to propose a method that can rapidly predict the inference latencies of DNNs on heterogeneous platforms.

C. Importance of Hardware Resource Modeling

In Sniper [21], they propose a self-update cloud–edge collaborative inference scheduling system with excellent prediction results and scheduling performance. However, Sniper directly uses hardware architecture parameters to characterize the computing power of devices without modeling analysis. We found in further experiments that its scalability becomes limited by the lack of platform modeling. When the bundled software framework is changed for the three platforms, as shown in Table I, the average error of DNN inference latency prediction fluctuates wildly.

TABLE I
DIFFERENT SOFTWARE FRAMEWORK OF PLATFORMS

Platform	CUDA	PyTorch	NumPy
NVIDIA AGX	10.0	1.4	1.19
NVIDIA TX2	10.2	1.8	1.19
NVIDIA NX	10.2	1.6	1.13

TABLE II
DNN INFERENCE LATENCY PREDICTION FOR SNIPER ON HETEROGENEOUS PLATFORMS

MODEL	NVIDIA AGX			NVIDIA TX2			NVIDIA NX		
	Predict (ms)	Test (ms)	MAPE (%)	Predict (ms)	Test (ms)	MAPE (%)	Predict (ms)	Test (ms)	MAPE (%)
densenet121	0.223	0.256	15%	0.437	0.322	26%	0.371	0.344	7%
densenet169	0.280	0.341	21%	0.538	0.434	19%	0.465	0.466	0%
densenet201	0.376	0.436	16%	0.699	0.547	22%	0.609	0.585	4%
resnet152	0.587	0.825	41%	0.821	0.923	12%	0.992	0.956	4%
resnet101	0.412	0.535	30%	0.580	0.599	3%	0.697	0.621	11%
resnet50	0.245	0.171	30%	0.353	0.203	43%	0.413	0.213	48%
vgg16	0.407	0.685	69%	0.691	0.685	1%	0.707	0.685	3%
vgg19	0.481	0.624	30%	0.796	0.624	22%	0.833	0.624	25%
inceptionv3	0.195	0.332	70%	0.289	0.332	15%	0.331	0.332	0%
Average			36%			18%			11%

In Table II, the mean absolute percentage error (MAPE) for the three platforms are 36%, 18%, and 11%. Experiments show that the HRM method relying only on hardware parameters cannot represent the performance differences due to different versions of software frameworks. More seriously, due to the architecture of accelerators varying considerably between different manufacturers and different versions, such as GPUs and NPUs, it is not realistic to apply hardware parameters to achieve a unified representation of heterogeneous platforms. This is further exacerbated by the fact that most companies do not disclose detailed information regarding their hardware architecture development.

Observation 3: The modeling ability of hardware parameter is insufficient and cannot achieve performance model for highly heterogeneous platform. It is necessary to find a unified HRM method to make the DNN inference scheduling framework possess higher scalability on heterogeneous platforms with better scheduling performance and prediction effect.

In summary, obtaining DNN inference latency in advance can significantly enhance scheduling efficiency. However, the rapid development of DNNs and smart devices results in deployment overhead for latency measurement. In Sniper, changing the software framework of a platform leads to unstable predictions, which impacts scalability across heterogeneous platforms. Therefore, it is crucial for scheduling frameworks to propose a unified modeling approach for platforms in DNN latency prediction, considering the hardware architecture and its bundled software framework as a complete platform [28].

III. SYSTEM MODEL

To solve the scheduling challenges mentioned in the motivation, this section defines the task scheduling and DNN

latency prediction problems in the heterogeneous inference system.

A. Scheduling Problem Statement

The system $\mathbb{H}$ is equipped with several types of intelligent computing platforms, i.e., $\mathbb{H} = \{h_1, h_2, \ldots, h_m\}$, where h_i is a variety of heterogeneous platforms. Within a heterogeneous computing cluster, each node consists of the platform equipped with different acceleration chips, so computing heterogeneity and communication heterogeneity are common among different nodes. Meanwhile, with the diversity of accepted DNN tasks, the computing cluster needs to schedule DNN tasks based on their performance characterization on each node and evaluate the scheduling algorithm performance by the throughput and the average waiting time.

In this scheduling problem, we define the DNN task set ($\mathbb{X}$) to consist of model (M), data (D), and maximum response time (Tm), i.e., $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$, $x_i = \{M_i, D_i, Tm_i\}$. In the description, the model M_i is the directed acyclic graph of the DNN from which the network features can be extracted, and the data D_i determines the number of times a task needs to respond to completion. Therefore, the actual response time from task i scheduling to node j can be described as

$$Tr_i = Tw_i + Td_{i,j} + Ti_{i,j} + Tb_j \tag{1}$$

where Tr_i is the actual response time of task i after scheduling; Tw_i is the waiting time of task i; $Td_{i,j}$ is the time consumed for the dataset D_i to be transmitted to node j, i.e., the transmission delay; $Ti_{i,j}$ is the inference time of task i on node j; and Tb_j is the time consumed for node j to transmit the inference results back.

According to the above analysis, the objective function of this scheduling problem is to minimize the task response time while ensuring that each task completes before its own maximum response time, which can be expressed as

$$\begin{aligned} &\min \sum_{i=1}^{n} Tr_i \\ &\text{s.t.} \quad Tr_i < Tm_i, \quad i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{2}$$

B. Prediction Problem Statement

In this article, we focus on inference latency prediction of artificial intelligence (AI) tasks on nodes to optimize the overall cluster throughput and QoS by inference latency awareness. Our goal is to design a prediction model that can accurately predict the inference latency of unseen DNNs on various platforms.

With the definition in Section III-A, we define a task dataset $\xi = \{H_\xi, X_\xi, Y_\xi\}$, where $H_\xi \subset \mathbb{H}$ is a set of platforms, $X_\xi \subset \mathbb{X}$ is a set of DNNs, and $Y_\xi \subset \mathbb{Y}$ is a set of latencies of X_ξ directly measured on each platform H_ξ. Since the measured latency is dependent on both the platform type (h_i) and the network architecture (x_i), the hardware-conditioned prediction model is proposed as follows:

$$f(Vx_i, Vh_i; \theta) : \mathbb{X} \times \mathbb{H} \to \mathbb{R} \tag{3}$$

where Vx_i is the feature of the network architecture (x_i), Vh_i is the feature of the platform type (h_i), and $f(Vx_i, Vh_i; \theta)$ is the inference latency of network (x_i) on platform (h_i).

Equation (3) can predict the latency differently depending on the platform type h_i and the network x_i. A crucial challenge of our hardware-conditioned prediction model is how to represent the neural network feature Vx_i and the platform feature Vh_i uniformly, which is described in Section IV-B.

Then, our goal is to learn a model $f(Vx_i, Vh_i; \theta)$ parameterized by θ that estimates the latency $y \in Y_\xi$ for each given platform h_i by minimizing empirical loss $\mathbb{L}$ (e.g., mean-squared error) on the predicted values $f(X_\xi, H_\xi; \theta)$ and actual measurements Y_ξ as follows:

$$\min_{\theta} \mathbb{L}\big(f\big(X_\xi, H_\xi; \theta\big), Y_\xi\big). \tag{4}$$

The core of completing the above model design depends on the characterization of DNNs (X_ξ) and platform (H_ξ), as well as the design of the model ($f(\cdot)$). *Ace-Sniper* describes this part of the work in detail in the introduction to core modeling methods (Section IV-B), including the task modeling method, the HRM method, and the PCN design.

IV. Framework Design

This section introduces *Ace-Sniper* with DNN inference latency modeling on heterogeneous device. Section IV-A shows the overview design of *Ace-Sniper*, Section IV-B details the three core modeling methods, and Sections IV-C and IV-D introduce the offline training phase and online operation phase.

A. Overview Design

Fig. 3 depicts the design of *Ace-Sniper* in cloud–edge collaborative scheduling. The framework consists of an offline training phase and an online operation phase. To tackle the challenge of uniform modeling across highly heterogeneous platforms, a new method called HRM is proposed by considering platforms as black boxes and outputs feature vectors. To comprehensively characterize the platform heterogeneity in HRM, we introduce a new dimension (inference data flow) for the network structure analyzer and enrich the parameterized model of the random network generator. As a result of the introduction of HRM, the inputs of PCN are modified and the network structure needs to be adapted.

The offline training phase is mainly responsible for training PCN and generating random network data. In this phase, a large number of random networks and inference latencies on each platform are obtained through the random network generator. Utilizing these measurements, the framework considers the platform as a black box that outputs feature vectors to enable HRM of heterogeneous platforms. Subsequently, PCN training is performed using the results of HRM and DNN features to converge and ultimately achieve predictable network inference latencies.

The online operation phase mainly focuses on task feature modeling and inference latency prediction in task scheduling. During this phase, the framework utilizes the network structure analyzer to analyze the constantly iterating and updating DNN network structure. The structure checker evaluates

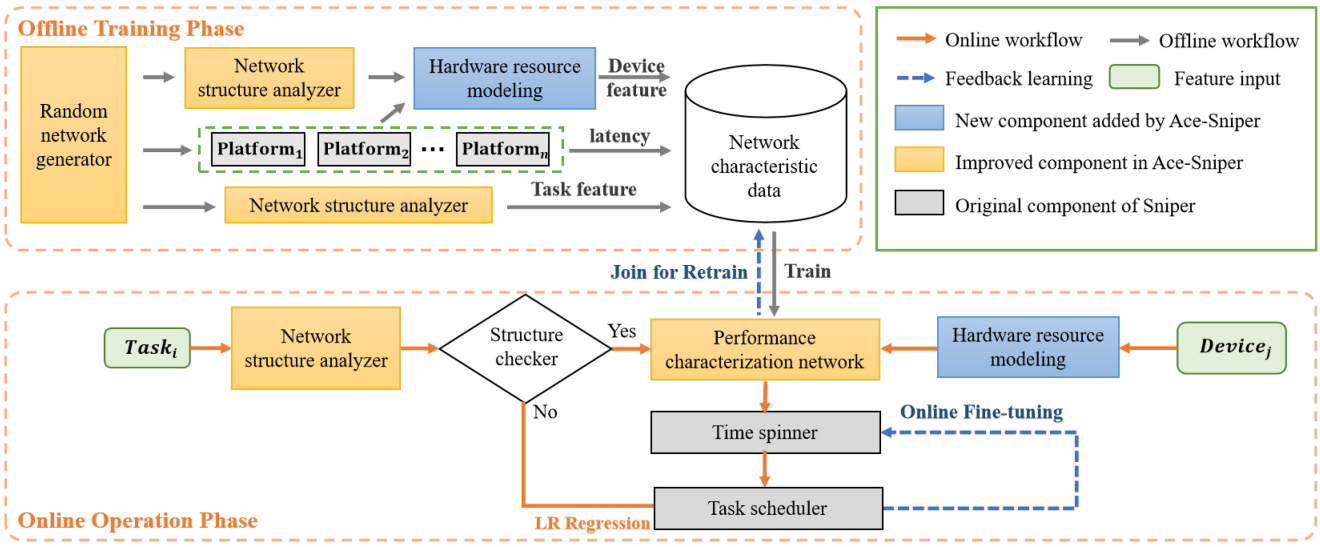

Fig. 3. Overview of the scheduling framework (*Ace-Sniper*) with DNN inference latency modeling on heterogeneous device. In *Ace-Sniper*, the core components include: random network generator, network structure analyzer, HRM, PCN, and task scheduler; the supporting component is time spinner.

the DNNs that require performance prediction, while the remaining DNNs proceed to the task scheduler directly. The PCN predicts the inference latency based on the result of HRM and task modeling obtained during the offline training phase. To improve prediction accuracy, the time spinner is utilized to fine-tune the results. Finally, the task scheduler correctly schedules each task based on the predicted inference latency, improving the QoS of the tasks and the TPS of the cluster.

Next, we present the details based on three parts: 1) the core modeling methods; 2) the offline training phase; and 3) the online operation phase.

B. Core Modeling Methods

Task Modeling Method: The task modeling method is mainly performed by the network structure analyzer, which performs offline analysis of the neural network and extracts network structure features relevant to performance prediction. The network structure features (Vx) are the characterization of DNNs as follows:

$$Vx_i = [p_{i,1}, p_{i,2}, \ldots, p_{i,n}] \quad (x_i \in X_\xi) \tag{5}$$

where p_i is the element that can represent the task characteristics, as shown in Table III.

In the design of the network structure analyzer, a feature extraction method for the constantly iterative and self-updating models is required to provide sufficient inferential behavioral features for network performance prediction. It is mentioned in [29] that several types of smart acceleration chips accelerate fundamental network layers in various fashions. Notably, the inference behavior of fundamental network layers is similar across the same type of platforms. Based on this finding, the network behavior features can be evaluated in six dimensions mentioned in Fig. 4, which are shown in Table III. To comprehensively assess the performance of various DNNs on highly heterogeneous platforms, a new dimension called inference data flow is introduced, which primarily focuses on the size of the feature map during DNN inference. Inference data flow can be adjusted with the expanded random network generator during training to complete dataset construction and acquire more network features for predicting inference latencies during scheduling.

TABLE III
DNN FEATURE EXTRACTION BASED ON SIX DIMENSIONS

Dimensions	Feature
Network Structure	maximum number of branches; number of layers; number of critical path layers; proportion of critical path parameters, layers and computation; average computation of network layers
Calculation Times	number of computation; number of critical path; computation; number of convolutional computation; number of fully connected computation
Number of Parameters	number of parameters; number of critical path parameters; number of convolutional parameters; number of fully connected parameters
Proportion of Network Layer	number and proportion of normalized layers, convolutional layers (1x1, 3x3, 5x5, 7x7), atrous convolutional layers (3x3), fully connected network layers, concat layers, add layers, pooling layers (average, maximum)
Inference Data Flow	number of feature map planes (14, 28, 56, 112, 224) and channels (3, 32, 64, 128, 256, 512)
Dependency	Determined by actual task performance

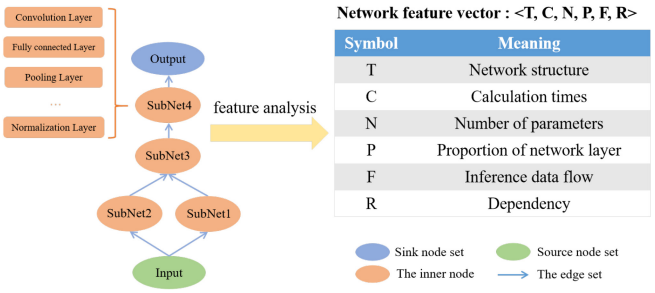

Fig. 4. Modeling analysis of DNN through six dimensions.

Algorithm 1 Key Feature Extraction

Input: F_M: features matrix, T: latency, K: max length of K_F
Output: K_F: key features

```
function KFE(F_M, T)
    Load F_M and T, Initialize K_F by NULL
    while length of K_F less than K do
        Calculate Pearson coefficient of each feature in F_M with respect to T
        Select the feature with the largest Pearson coefficient into K_F
        Delete the selected feature from F_M
        Calculate the residual error (E) of each feature in K_F after linear fitting
        T ← E
    end while
end function
```

To extract critical performance-related features from neural networks, the random network generator is used to build a number of DNN performance datasets on the platforms (GPUs and NPUs). Using the extracted features, *Pearson coefficients* [30] are utilized to determine the correlation between each feature and the measured inference latency. The linear residual fitting method is utilized to continue the critical feature extraction process based on the calculated *Pearson coefficients*. The procedure of extracting the relevant features is illustrated in Algorithm 1.

Through the multistage feature processing above, a total of seven key features are extracted from Table III, including total computation, the number of atrous convolution layers (3×3), the number of convolution layers (1×1, 5×5), the number of critical path network parameters, the proportion of critical path network parameters, and the number of feature map planes (112). Based on the above features, we predict the network performances using a traditional fitting algorithm (Bayesian ridge regression) on a single platform. On the GPUs (NVIDIA AGX, TX2, and P4) and NPUs (Cambricon MLU220 and MLU270), their average prediction errors of the training and test sets are within 0.67 and 2.8 ms, respectively. These results demonstrate that the above features can achieve performance feature fitting of most networks on a single platform.

HRM Method: HRM is mainly accomplished to characterize the computing power of heterogeneous platforms, which includes general and specific performance characterization. The performance characterization (Vh_k) is the characterization of the platforms as follows:

$$\begin{aligned} Vh_k &= \left[Hg_k, Hs_k\right] \\ &= \left[w_{k,1}, w_{k,2}, \ldots, w_{k,n}, y_{k,x_1}, y_{k,x_2}, \ldots, y_{k,x_t}\right] \left(h_k \in H_\xi\right) \end{aligned} \tag{6}$$

where $w_{k,i}$ is the element that can represent the general performance characterization (Hg_h) of platform (h_k), and y_{k,x_i} is the element that can represent the special performance characterization (Hs_k) of platform (h_k).

One challenge in predicting DNN latency lies in developing a unified characterization method for the computing power of highly heterogeneous platforms (h_*). To tackle this challenge, we focus on analyzing the inference latencies of different DNNs on heterogeneous platforms. As shown in Fig. 5, it is possible for different DNNs to exhibit completely opposite relative performances on heterogeneous devices due to differences in hardware architectures, inference frameworks, etc. For example, the relative latencies of different DNNs mostly vary greatly between MLU270 and AGX, i.e., they do not have a linear variation relationship. These observations highlight the need for the HRM approach that can accurately capture the architectural features of devices with similar architectures and the inference latency differences among heterogeneous hardware when running different DNNs.

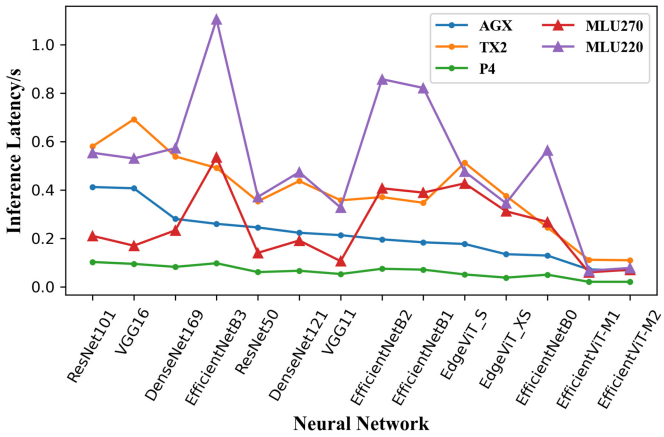

Fig. 5. Inference latencies of DNNs among different platforms.

Unlike the hardware architecture-aware modeling method, HRM considers the platforms as black-box functions that output feature vectors by using the computing performance of the platforms as the primary basis. To obtain a more comprehensive characterization of DNN inference latency on hardware, we leverage the fitted parameters of normalized inference latencies as an integral part of the feature vector, which allows us to capture the similarity of hardware architecture. Additionally, we employ some specific DNNs to effectively capture and characterize the variations in inference latencies among heterogeneous devices. The modeling process is described in detail next.

Initially, the latencies of the platforms on a fixed set of neural networks are obtained as follows:

$$Mh_k = \left[y_{k,x_1}, y_{k,x_2}, \ldots, y_{k,x_d}\right] \quad \left(h_k \in H_\xi\right) \tag{7}$$

where y_{k,x_i} is the inference latency of the reference networks (x_i) on the platform (h_k), x_i is the sample of the reference neural networks ($\{x1, x2, \ldots, xd\} \subset \mathbb{X}$), fixed across all tasks for each platform, and d is the number of networks. As for the reference architectures, we randomly sample them from the random network generator and set $d = 1000$ to cover various architectures and sizes of neural networks.

Based on the collected latency vector (Mh_k), the neural network inference latency prediction is viewed as a distribution mapping, i.e., $f(Vx_i, Vh_k; \theta)$. Vh_k is the feature vector of the platform (h_k), which consists of a general performance

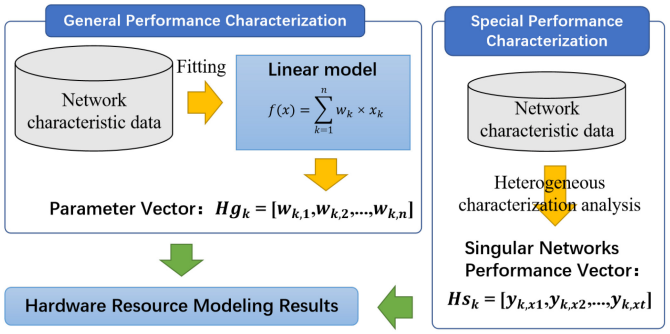

Fig. 6. HRM method.

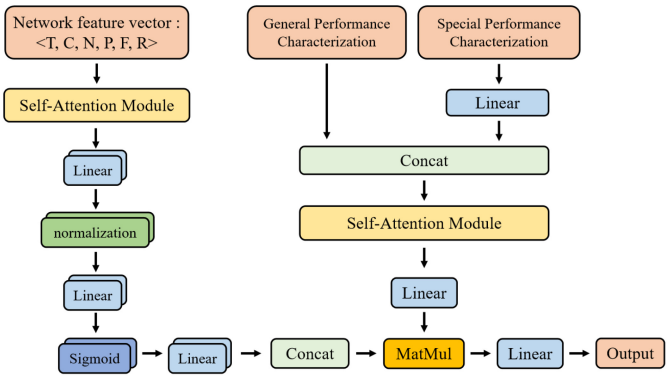

Fig. 7. PCN.

characterization and a special performance characterization, as shown in Fig. 6.

The general performance characterization (Hg_k) focuses on the acceleration efficiency of the platform in most DNNs inference, which takes the form of a parameter vector for the model used to fit the inference latencies. The linear model $gh_k(\cdot)$ is used to perform feature fitting on Mh_k as follows:

$$gh_k(Vx_i; w_k) \rightarrow y_{k,x_i} \quad \left(x_i \in \mathbb{X}, h_k \in H_\xi\right) \tag{8}$$

where parameters w_k is used for the linear fit of Vx_i to y_{k,x_i}.

Among them, the heterogeneity among platforms which is expressed as a parameter vector of the fitted model as follows:

$$Hg_k = \left[w_{k,1}, w_{k,2}, \ldots, w_{k,n}\right] \quad \left(h_k \in H_\xi\right). \tag{9}$$

The special performance characterization (Hs) focuses on characterizing the degree of heterogeneity among platforms through the neural network performance differences. As a complement to the general performance characterization, we mainly use the inference latencies vector of *singular networks* to representation. The *singular networks* ($X_\kappa \subset \mathbb{X}$) are defined as networks that can better describe the computing power differences between heterogeneous platforms, which can be obtained from filtering the covariance of the inference latency matrix of the network across heterogeneous platforms. So, the general performance characterization vector (Hs) can be described as

$$Hs_k = \left\{y_{k,x_1}, y_{k,x_2}, \ldots, y_{k,x_t}\right\}\left(h_k \in H_\xi, x_i \in X_\kappa\right). \tag{10}$$

Finally, the computing power of the platform is characterized by using the general and special performance characterization, i.e., $Vh_k = [Hg_k, Hs_k]$.

PCN Design: PCN is primarily used to predict the performance of networks on heterogeneous platforms based on network features, i.e., $f(Vx, Vh; \theta) : \mathbb{X} \times \mathbb{H} \rightarrow \mathbb{R}$. In this part, we describe the concept of NNS and then use the PCN with the self-attention [31] module as a representation model for NNS to predict the performance of DNNs, i.e., $f(\cdot)$.

To analyze the differences in performance characterization of each network on edge platforms, this section introduces the concept of NNS, which is the similarity of the performance characterization of DNNs on a single platform based on its structure, behavior, and other characteristics. The proof of NNS for a single platform is shown in Section IV-B, and the DNN features can be used as the result of NNS modeling. However, the key features that affect DNN inference latencies will vary with heterogeneous platforms. Therefore, the results of HRM (Vh) are introduced to characterize the heterogeneous platforms and combined with NNS modeling to predict DNN inference latencies.

Additionally, as confirmed in Section IV-B, the relative performance between different tasks is not always nearly the same or even opposite across the highly heterogeneous platforms. Furthermore, different network features (network layers and operators) have different effects on the inference latency, and different platforms also map different latency distributions. Therefore, when considering multiple DNN inference latencies on a single platform, the inference latency prediction of DNNs on heterogeneous platforms can be viewed as the result of different mappings over various distributions. Prior to makes predictions, Mh^* is obtained by normalizing the latency vector (Mh) of the random network generator as follows:

$$\begin{aligned} M_h^* &= \left\{y_{k,x_1}^*, y_{k,x_2}^*, \ldots, y_{k,x_d}^*\right\} \quad \left(h_k \in H_\xi\right) \\ y_{k,x_i}^* &= \frac{y_{k,x_i} - \min(Mh_k)}{\max(Mh_k) - \min(Mh_k)}. \end{aligned} \tag{11}$$

Therefore, the PCN is designed with a self-attention module to characterize the results of NNS and HRM modeling. As the hardware-conditioned prediction model $f(Vx, Vh; \theta)$, PCN takes the network features (Vx) and hardware features (Vh) as input, and takes inference latency (Mh) as output. The network structure of PCN is shown in Fig. 7.

To increase the universality of the neural network features in Table III, the self-attention module is utilized to automate the combination of network features to achieve network key feature extraction [31]. Automatic feature derivation is accomplished in the self-attention module via the point product combination of network features, which generates the queue, key, and value in the self-attention module. Finally, the self-attention module is constructed by connecting the fully linked layer to the scaled point product attention. The self-attention of x can be calculated through the following equation:

$$\text{Attention}(x) = \text{softmax}\left(\frac{Q(x) \times K(x)^T}{\sqrt{d_k}}\right) \times V(x) \tag{12}$$

where $Q()$, $K()$, and $V()$ are the results for the query, key, and values of x, which are all vectors; and $[1/(\sqrt{d_k})]$ is the scaling factor.

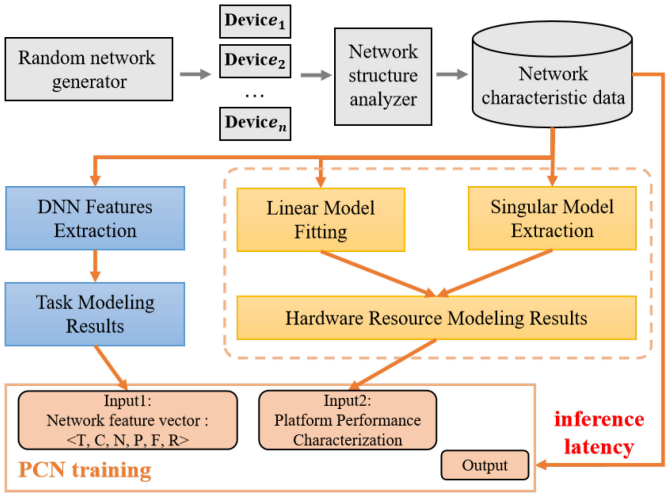

Fig. 8. Offline training phase of *Ace-Sniper*.

The network structure of PCN is illustrated in Fig. 7. With the addition of HRM, the PCN is required for processing two hardware features: 1) general performance characterization and 2) special performance characterization. The former is a parameter vector of fixed length that can be directly input to the self-attention module, while the latter is a vector consisting of inference latencies of individual networks on heterogeneous platforms. To extract deeper platform performance features, the PCN utilizes a fully connected layer to increase the dimensionality of the latency vector. This processed vector is then concatenated with the general performance representation vector and input to the self-attention module. Meanwhile, the neural network features include the features listed in Table III and are cross-combined with the self-attention module features. The output features from the self-attention module are connected to a multilayer fully connected network. A sigmoid layer is added to map features into the range [–1, 1] in preparation for multiplication with the critical features from HRM. Finally, the PCN completes its calculation, and the output of inference latencies is obtained. The calculation formulas for PCN are as follows:

$$\mathcal{F}(x) = \text{Concat}(\text{multi\_connect}(x)) \tag{13}$$

$$\text{Output} = \mathcal{F}(\text{Attention}(x_n)) \times \text{Attention}(x_d) \tag{14}$$

where multi_connect(x) is the multilayer fully connected network through which the self-attention output passes; concat(m_x) is a concatenation of the output (m_x) of a multiheaded network; x_n and x_d are the input for network feature and platform feature; and Output is the predicted inference latency of the network on the platform.

C. Offline Training Phase

The offline training phase needs to be executed before the framework is used, which pretrains the PCN by the random network data. As shown in Fig. 8, the random network generator generates a large number of random networks and saves their task modeling features (Vx). Then, these random networks are inferred on each heterogeneous platform in order to obtain inference latencies (Mh) and keep them in the database. HRM obtains the platform performance characteristics (Vh) based on Vx and Mh from platform h. The pretraining of the PCN is completed according to the formula

$$y_{x_i,h_k} = f(Vx_i, Vh_k; \theta) \quad (Vx_i \in \mathbb{X}, Vh_k \in \mathbb{H}) \tag{15}$$

where y_{x_i,h_k} is the inference latencies of the task (x_i) on the platform (h_k) in Mh.

Since both the network structure analyzer and HRM have been described in Section IV-B, the random network generator is the focus of this section.

The random network generator plays a crucial role in generating a vast number of random networks, which provide the necessary data support for the training of the HRM and PCN. As mentioned in Section IV-B, PCN has numerous parameters that require training on large volumes of measurement data. While the PCN encounters various types of networks during its operation, these networks ultimately have similar network layers and operators. This observation motivates us to design a parametrized model capable of permuting network layers and operators, which can be used to simulate the actual networks encountered during PCN operation.

The design of the random network structure contains operators, such as convolutional layers, atrous convolutional layers, pooling layers, normalization layers, and more. These operators are the most commonly used basic operators in modern DNN design and account for the majority of the computational overhead of DNNs. Therefore, different DNNs can be imitated with the customized basic blocks. As shown in Table IV, the parameterized model has nine adjustable architectural parameters, which constitute a huge design space. By analyzing various DNNs in real scenarios, we design a reasonable range for each parameter that can be tuned. This range is slightly larger than the existing DNN design space, making the random network generator applicable for highly heterogeneous platforms.

TABLE IV
TUNING RANGE OF THE PARAMETERIZED MODEL

Parameter Range	Range
Input planes	[14, 28, 56, 112, 224]
Output planes	[14, 28, 56, 112, 224]
Input channels	[3, 32, 64, 128, 256, 512]
Output channels	[32, 64, 128, 256, 512]
Network type	[conv, pool, norm, full]
Kernel size	[1, 3, 5, 7]
Stride	[1, 2]
branches	[1, 2, 3]
Connection method	[None, Add, Concat]

D. Online Operation Phase

Fig. 3 contains an illustration of the online operation phase, which involves extracting features from arriving DNN networks (Vx) and combining them with known platform features (Vh) to form input for the PCN. PCN predicts the DNN inference latency on each heterogeneous platform, and the task scheduler performs task scheduling based on the

latency prediction to enhance the throughput and QoS of the heterogeneous computing platform.

The next focus is on three components: 1) structure checker; 2) time spinner; and 3) task scheduler. The other related components have been described in detail in Section IV-B.

Structure Checker: Structure Checker is primarily responsible for selecting a prediction strategy based on the DNN features. Structure Checker uses the overall computational volume to determine the size of the network. The medium or large networks flow to the PCN for prediction. And the latencies of small networks are predicted by traditional prediction models.

Time Spinner: Time Spinner is mainly utilized in online fine-tuning. It can select the network features that best fit the prediction residuals and make the prediction results more accurate by parameter fine-tuning. To achieve fine-tuning of *Ace-Sniper*, the *Pearson coefficient* of each feature is calculated by using the predicted residuals as an indicator, and the feature with the highest correlation is selected to residual fitting.

Task Scheduler: The task scheduler is mainly responsible for generating scheduling decisions based on the scheduling matrix to minimize task waiting time with a guaranteed TPS. Finding the optimal scheduling decisions for multiple DNN tasks on various platforms proved to be an NP-Hard problem. However, in the *Sniper*, this scheduling problem can be obtained by the relaxation model *P* based on the generated scheduling matrix. Current algorithms to solve such problems include least laxity first (LLF) [32], heuristic algorithm [33], etc. To quickly obtain the optimal scheduling strategy, *Ace-Sniper* schedules the tasks using the LLF with extra limits, which ensures the maximum TPS while reducing waiting time.

V. Experiment and Evaluation

This section focuses on evaluating the prediction accuracy of PCN and the scheduling performance of *Ace-Sniper* in multiple heterogeneous platforms and DNNs. Besides, compared to the actual measurement method, we evaluate the performance advantage and lower system cost of *Ace-Sniper* when the new iteration networks arrive.

A. Experimental Setup

Hardware Platforms: In order to realize the scenario with fully heterogeneous hardware, this experimental environment is based on a heterogeneous computing cluster consisting of ten nodes, including three types of GPUs (NVIDIA AGX, TX2, and P4) and two types of NPUs (Cambricon MLU220 and MLU270). Among them, NVIDIA AGX, TX2, and MLU220 are directly acquired developer kits and belong to edge platforms; and NVIDIA P4 and MLU270 are mounted on edge servers and belong to low-power computing boards.

We select 2500 models from the random network generator as the test set and as part of the inference task set. In addition, the ordinary networks include *DenseNet121, DenseNet169, EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, ResNet50, ResNet101, VGG11, VGG16, EdgeViT-XS, EdgeViT-S, EfficientViT-M1, EfficientViT-M2,* and *MobileViT-S*. The training set of PCN consists of 23 000 samples in the random network generator to ensure stable convergence.

TABLE V
Prediction Results of Ordinary Neural Networks on Heterogeneous Platforms

Network	MAPE(%)				
	AGX	TX2	MLU220	MLU270	P4
DenseNet121	14.58	11.13	6.81	5.94	1.64
DenseNet169	9.60	14.67	5.76	5.27	0.5
EfficientNetB0	27.21	3.92	14.72	6.95	14.2
EfficientNetB1	9.24	10.63	3.41	6.42	25.21
EfficientNetB2	12.46	7.53	5.03	8.75	21.76
EfficientNetB3	5.01	14.9	17.33	23.16	26.05
ResNet50	16.68	21.08	18.72	13.03	22.95
ResNet101	4.89	7.88	7.83	1.34	15.08
VGG11	24.3	9.29	11.31	21.58	1.44
VGG16	14.64	21.96	24.18	3.66	20.04
EfficientViT-M2	4.31	10.95	14.24	9.77	6.93
EfficientViT-M1	9.59	9.63	13.75	3.09	5.49
EdgeViT-XS	5.17	9.88	5.32	18.46	4.48
EdgeViT-S	7.80	7.98	3.95	2.15	4.83
MobileViT-S	13.53	17.08	-	-	3.31
Average	11.93	11.90	10.88	9.26	11.59

Prediction Comparison Baselines: This article focuses on the noninvasive DNN performance prediction methods, and propose the predictor (PCN) that consists of the HRM and NNS modeling. According to the survey, the current work widely adopts *FLOPs* [19] and *FLOPs&AMC* [18] for task feature modeling, which use FLOPs and memory access cost (i.e., AMC) to estimate model latency. A recent work [22] mentions an HRM method (*HELP*), which uses the inference latency vectors of several random neural networks to characterize the heterogeneous platforms. For comparison with our proposed task modeling and HRM ((6) NNS+HRM), five related methods are evaluated: 1) *FLOPs+HELP* [19], [22]; 2) *FLOPs&AMC+HELP* [18], [22]; 3) NNS+*HELP* (Sniper) [21], [22]; 4) *FLOPs*+HRM [19]; and 5) *FLOPs&AMC*+HRM [18].

In the prediction experiments, the test datasets include random neural networks and ordinary neural networks in Table V. We evaluate the prediction performance by the MAPE, which is the standard metric in regression. Besides, the ±5%, ±10%, and ±15% accuracy [34] are introduced as important metrics, that are the percentage of models with predicted latency within the corresponding error range.

Scheduling Comparison Baselines: The scheduling framework (*Ace-Sniper*) is evaluated on the computing platform described in Section V-A. The comparison experiment chooses 1) *Polling* [12] and 2) *EDF* [11] mentioned in motivation as the baselines, and reproduce the scheduling experiment 3) (*SLO-Scheduler*) in [10] as the performance ceiling, which uses the measured task latencies as the performance awareness. (*without considering the real measured cost of the DNN tasks*)

The task scheduling experiments construct a multinode heterogeneous computing platform built from the five types of platforms and used the neural networks mentioned in Table V as the computing tasks of the platform. In addition, to simulate real-world request generation scenarios, we synthesize the requests under various generation scenarios [35], [36] and use

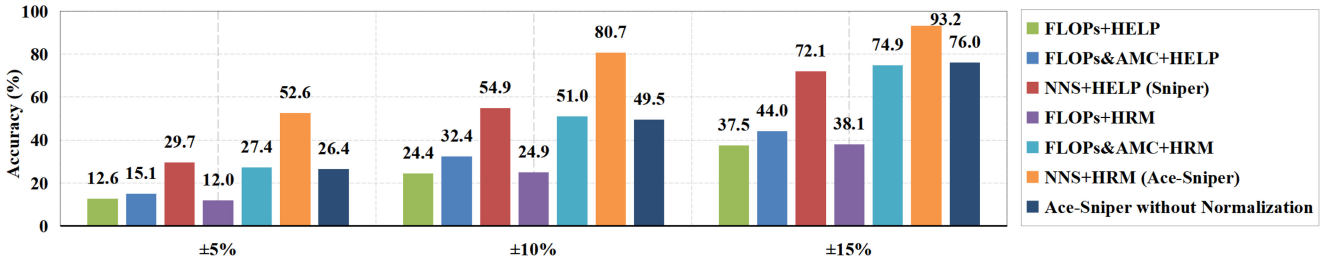

Fig. 9. Compared to baseline predictors, *Ace-Sniper* achieves much higher accuracy on random neural networks (±5%, ±10%, and ±15%).

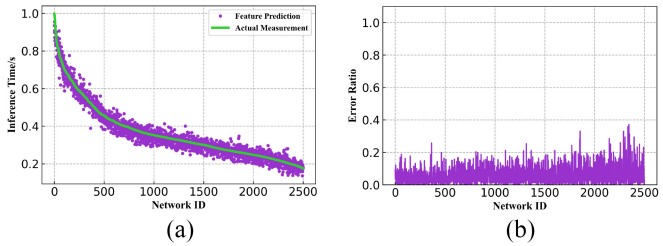

Fig. 10. Prediction results of PCN for random networks. (a) Comparison of predicted results. (b) Error ratio of prediction.

TABLE VI
PREDICTION RELATIVE ERRORS OF RANDOM NETWORKS ON PLATFORMS

Scope	MAPE(%)					
	AGX	TX2	MLU220	MLU270	P4	ALL
0-25%	7.7%	7.1%	10.8%	10.6%	4.8%	7.8%
25-50%	5.3%	5.8%	7%	7.1%	3.6%	5.8%
50-75%	5.4%	5.1%	6.6%	6.8%	3.5%	5.5%
75-100%	5.4%	4.4%	4.2%	4.5%	4.1%	4.7%

Poisson distribution to simulate the task generation rate, which can be adjusted by the frequency and distribution of request arrivals according to the experiment requirements. Finally, the superiority of the algorithm is evaluated based on the throughput of the inference platform and the average waiting time of tasks.

B. DNN Performance Prediction

This section focuses on testing the performance prediction error of PCN. As described in Section IV-C, the trained PCN is used to predict the model inference latencies on heterogeneous platforms and compare with baselines.

Predicted Results: Fig. 10(a) shows the prediction accuracy on a custom random network. For the sake of observation, we sort the random networks according to their inference latencies and assign network IDs in order. As shown in Fig. 10(b), the relative errors of most network predictions are small, with an average relative error of only 6.10%, and the proportion within ±15% accuracy is 93.2%. Moreover, most error values do not exceed 0.1 s, and the average error across multiple platforms remains at 0.021 s. Immediately after, the feasibility of PCN is verified by empirically testing the actual DNNs with different structures. As shown in Table V, the average relative error of actual network prediction is about 11.11%, which is in line with our expectations. It should be noted that inference testing of *MobileViT-S* is unavailable due to the lack of support for operators with large feature maps on MLU270 and MLU220.

Comparison With Baselines: As described in Section IV-B, NNS task modeling and HRM are proposed as inputs of PCN for predicting DNN inference latency in *Ace-Sniper*. As a comparison, Fig. 9 displays the prediction accuracy achieved by predictors composed of different modeling methods.

Compared to the baseline, only *Ace-Sniper* achieves accurate DNN latency prediction among highly heterogeneous platforms, with a prediction accuracy 21.1% higher than Sniper using the ±15% accuracy metric, thanks to the strength of HRM in characterizing platforms.

In this section, we conduct a comparative analysis of prediction accuracy using the ±15% accuracy metric. Without HRM, the NNS+*HELP* predictor with NNS task modeling achieves 72.1% accuracy, which is significantly better than *FLOPs+HELP* (37.5%) and *FLOPs&AMC+HELP* (44.0%). This reflects the advantages of NNS over other task modeling approaches. With HRM, the *Ace-Sniper* (NNS+HRM) predictor achieved a prediction accuracy of 93.2%, significantly better than *FLOPs*+HRM (38.1%) and *FLOPs&AMC*+HRM (74.9%). And the advantages of HRM can be clearly demonstrated under the comparison of *FLOPs&AMC+HELP* with *FLOPs&AMC*+HRM. In addition, the NNS+HRM predictor also has a significant advantage at ±5% and ±10% accuracy. The final two sets of comparison experiments demonstrate the effectiveness of the inference latency normalization method for heterogeneous platforms, resulting in a 17.2% improvement in prediction accuracy.

Prediction Error Analysis: To analyze the prediction relative error of PCN, the test dataset of Fig. 10(b) is sorted in ascending order of measured latencies and divided into four groups according to the quartiles. The prediction relative bias of each group is shown in Table VI. On the one hand, the large relative errors mainly come from small latencies (i.e., EfficientNetB0, EfficientNetB1, and ResNet50 models $<$ 150 ms). This is due to the fact that smaller models have a lower measured inference latency, which results in a large relative bias even in the face of a small absolute bias. On the other hand, the large relative errors also come from Cambrian MLU220 and MLU270. This could be attributed to the fact that Cambrian is an emerging computing chip, which may experience performance degradation and instability due to the lack of support for certain operations during the online operation phase. This could potentially affect the measured latencies of DNNs and introduce extra prediction bias. Fortunately, the NVIDIA platforms (with excellent stability and operator

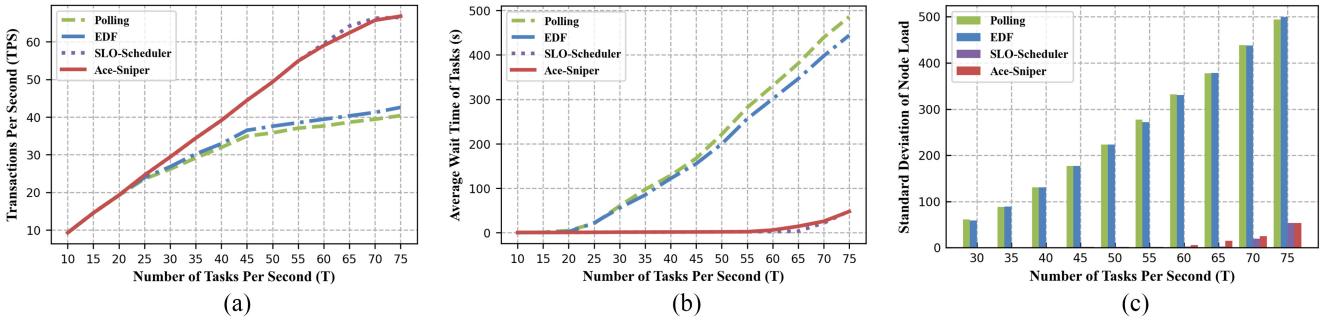

Fig. 11. Scheduling results of Polling, EDF, SLO-Scheduler, and *Ace-Sniper*. Polling and EDF denote the polling scheduling algorithm and EDF algorithm without performance awareness; SLO-Scheduler denotes the replication of the comparison experiment for [10]; *Ace-Sniper* denotes the scheduling experiment based on the noninvasive PCN; and T is the average number of tasks arriving per second. The duration of the experiment is 400 s, and the platform inferred the images with a batchsize of 16. (a) TPS of task. (b) Average waiting of task. (c) Standard deviations of nodes load.

support) all demonstrate superior prediction results across various software versions, despite the significant heterogeneity between NVIDIA P4 and TX2.

C. Schedulability Experiments

This section focuses on evaluating the performance of Ace-Sniper scheduler under different DNN task generation distributions, including performance and node load. As shown in Section IV-D, we compare the performance of the scheduling implemented in the online operation phase with the three baselines. To show the excellence of *Ace-Sniper* compared to the measurement-based SLO-Scheduler, we test the performance of each scheduling algorithm under different new iterative network probability and analyze the scheduling overhead of *Ace-Sniper* and SLO-Scheduler. In particular, *Ace-Sniper* employs the network structure analyzer to analyze the compatibility of DNNs with devices, effectively preventing inappropriate scheduling, such as assigning *MobileViT-S* to MLU270. To ensure fairness in scheduling evaluation, other scheduling algorithms incorporate manual settings to prevent the assignment of these DNNs to unsupported devices.

Performance Analysis: This experiment sets the TPS and the average wait time (AWT) as the observed quantities for evaluating the scheduling performance in the different number of tasks per second (T). The AWT can be used as a quantitative indicator of the QoS, and the TPS can be used as a reflection of throughput. The scheduling results of Polling, EDF, SLO-Scheduler, and *Ace-Sniper* are shown in Fig. 11, including TPS, AWT, and node load standard deviation.

Obviously, Fig. 11 demonstrates that *Ace-Sniper* performs similarly to SLO-Scheduler (T<55) in terms of scheduling performance with the prediction accuracy guarantee of PCN, which implements time-aware task-based scheduling.

Compared with the scheduling algorithm without time awareness (Polling and EDF), *Ace-Sniper* starts to have a significant advantage at T>25, and the advantage becomes more obvious as T increases. When T<55, the scheduling performance of *Ace-Sniper* and SLO-Scheduler is identical (AWT = 0), i.e., the task is executed immediately upon arrival. When $T = 55$, *Ace-Sniper* achieves 54.6 in TPS, which is better than Polling (49.1%) and EDF (41.1%), and achieves 0 s in AWT, which is less than Polling (291 s) and EDF (263 s). After T>55, *Ace-Sniper*'s AWT slowly increases and the TPS growth rate slows down, slightly lower in value than SLO-Scheduler, but performs consistently again at $T = 70$. When $T = 70$, *Ace-Sniper* has a significant reduction in AWT compared to both Polling (83.8%) and EDF (82.1%), and a significant increase in platform throughput (TPS) compared to Polling (66.6%) and EDF (59.4%).

In addition, the load balancing between the algorithms for the different T is evaluated by the standard deviations of node load [23]. As shown in Fig. 11(c), *Ace-Sniper* achieves the same performance as SLO-Scheduler in terms of load balancing in the load range ($T \leq 55$). Even though the load on *Ace-Sniper* increases with the increase of T increases, it still outperforms the scheduling algorithm without time awareness (Polling and EDF) and is not significantly different from SLO-Scheduler.

New Iterative Network: In the scheduling experiments, it can be observed that the measurement-based SLO-Scheduler slightly outperforms *Ace-Sniper* in terms of throughput and QoS when the measured scheduling overhead is not taken into account. However, due to the variable user requirements in actual operation, the actual neural network in operation will always be continuously optimized (e.g., pruning, quantization, etc.), resulting in the newer iterations of the neural network.

In response to the emergence of these new networks, the performance of measurement-based SLO-Scheduler may not be as good. The new networks do not have actual measurement data as scheduling support, which makes the measurement-based SLO-Scheduler degenerate into task scheduling algorithms without time awareness (e.g., polling, EDF, etc.).

Based on the experiments in Section V-C, we added the factor of new iterative network probability to further compare SLO-Scheduler and *Ace-Sniper*. The experimental results for $T = 65$ are shown in Fig. 12, with Fig. 12(a) showing the throughput analysis and Fig. 12(b) showing the average waiting time analysis. It can be seen that as the probability of a new iteration network increases, the performance of the SLO-Scheduler gradually deteriorates and finally degrades to polling. After the new iteration of the network is greater than 20%, *Ace-Sniper* starts to outperform the measurement-based scheduling algorithm (SLO-Scheduler).

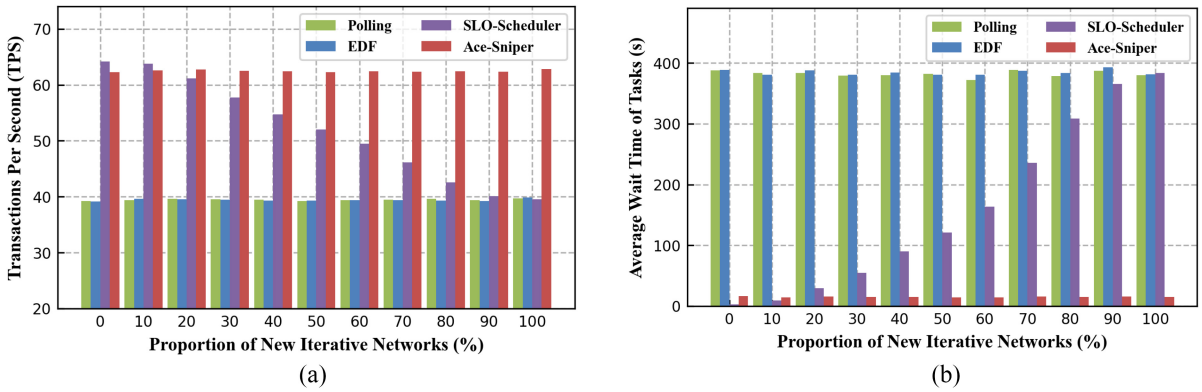

Fig. 12. Scheduling results on different proportions of new iterative networks. To ensure the randomness of task arrivals and the same amount of tasks, the tasks are randomly generated according to Poisson distribution and the number of tasks per second (T) is 65 in each experiment. At the same proportion of new iterative networks, the task arrival list is consistent. (a) TPS of task. (b) Average waiting of task.

TABLE VII
TIME COST OF ACE-SNIPER

Phase		AGX	TX2	MLU 220	MLU 270	P4
Pre-training	Training set generating	9hr	15hr	17hr	5hr	4hr
	PCN training	120s				
Feedback learning	*PCN* retraining	60s				
	Spinner training	10s				
Per schedule		0.05s				

Cost Analysis: The *Ace-Sniper* requires PCN pretraining before being used. The pretraining phase consists of two parts, including the training set generating and PCN training. As shown in Table VII, most of the cost comes from the training set generating, which can be linearly scaled down by increasing more platforms. In the process of framework application, the Time Spinner takes about 10 s to adjust the parameters in batches for a certain number of tasks accumulated, and the PCN takes 60 s for retraining. Both the pretraining and Feedback learning are executed when the platform is free and do not increase the latency per scheduling.

The time cost of the scheduling phase can be divided into three main parts: 1) inference latency acquisition; 2) scheduler computation; and 3) data transfer (DNNs and dataset). Among these, the difference in cost between *Ace-Sniper* and *SLO-Scheduler* is primarily due to the different approaches to obtaining inference latency, as the other components are nearly identical. When a large number of DNNs are updated and iterated simultaneously, *SLO-Scheduler* must send each DNN to each platform for serial inference latency measurement, with time cost increasing as inference latency increases. On the other hand, *Ace-Sniper* can quickly obtain the inference latency of all DNNs in parallel using cloud computing power, which stable performance at around 0.05 s for any DNN. In summary, the inference latency acquisition cost in *Ace-Sniper* is significantly lower than that in *SLO-Scheduler*.

VI. RELATED WORK

Performance Characterization for DNN: Several existing studies [7], [8] report on the performance and inference time of DNN by seeing the results on a test set, which results in an unnecessary waste of computational resources and oscillations in prediction accuracy. Further, recent research [37] has stressed the necessity of performance characterization methodologies for DNN training and inference acceleration. The work in [38] conducts a quantitative analysis of the runtime performance of various scheduling techniques. Additionally, the cost model in [17] and [20] contains static indications (e.g., the number of kernel launches, FLOPs, and memory utilization) and dynamic metrics that require specific hardware measurements. Unfortunately, accurate performance prediction is tricky in these works because of the absence of a standard DNN characterization approach and device heterogeneity awareness.

Meanwhile, one work successfully predicted the memory usage of each DNN layer, demonstrating the superiority of a noninvasive approach. However, the analysis of DNN inference latencies is more complex than memory analysis. Then, in [21], they predict DNN inference latencies by a noninvasive method based on NNS modeling. However, the method has limited scalability since only the essential hardware parameters of heterogeneous devices are considered. To predicate latency on heterogeneous devices, recent work has been proposed [22]. They obtained inference time vectors to characterize the hardware by deploying custom benchmarks, severely limiting their ability to characterize the hardware and resulting in low prediction accuracy.

Cloud–Edge Collaborative Inference: The standard methods [39], [40] for cloud–edge collaborative inference is to divide the DNN into two parts that operate in the cloud and at the edge. Another method [41] uses the intermediate layer's confidence level to determine whether the remaining layers should be executed in the cloud or at the edge. In terms of scheduling, a great number of studies optimize scheduling performance by utilizing real running inference time [9], [10]. Recently, inference scheduling in the cloud has been extensively investigated in representational work [7]. However, for

cloud–edge collaborative inference, the same enough computer resources are not available [27].

From all these works on performance characterization and inference scheduling, the unique contribution of *Ace-Sniper* is to explore performance characterization similarity of DNNs on heterogeneous devices through introducing the NNS concept and the HRM approach, which allows the system to determine the inference latencies intelligently.

VII. Conclusion

This article proposes *Ace-Sniper*, a scheduling framework with DNN inference latency modeling on heterogeneous devices, which supports rapid iterative DNNs and highly heterogeneous platforms. We describe in detail the core modeling methods in *Ace-Sniper*, including the DNN task modeling, the HRM, and the noninvasive PCN. Specifically, HRM builds a unified modeling method for different hardware architectures, combining three updated components to enable the extension of Sniper to highly heterogeneous platforms. The experimental results demonstrate that the proposed framework achieves accurate prediction of DNN inference latency on highly heterogeneous platforms and provides significant improvements in throughput and QoS for edge heterogeneous clusters. In our future work, we plan to introduce more heterogeneous platforms to improve the scalability of *Ace-Sniper*.

References

[1] 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. 58th ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Dec. 2021, pp. 409–414.

[2] "The KubeEdge SIG AI." Sedna. Website. 2020. [Online]. Available: https://github.com/kubeedge/sedna

[3] Linux Foundation. "Edgexfoundry." Website. 2018. [Online]. Available: https://www.edgexfoundry.org

[4] C. Caiazza, S. Giordano, V. Luconi, and A. Vecchio, "Edge computing vs centralized cloud: Impact of communication latency on the energy consumption of LTE terminal nodes," *Comput. Commun.*, vol. 194, pp. 213–225, Oct. 2022.

[5] 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.

[6] X. Zhang, M. Hu, J. Xia, T. Wei, M. Chen, and S. Hu, "Efficient federated learning for cloud-based AIoT applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2211–2223, Nov. 2021.

[7] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2019, pp. 392–405.

[8] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.

[9] X. Wu, H. Xu, and Y. Wang, "Irina: Accelerating DNN inference with efficient online scheduling," in *Proc. 4th Asia-Pacific Workshop Netw.*, 2020, pp. 36–43.

[10] W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park, "SLO-aware inference scheduler for heterogeneous processors in edge platforms," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, p. 43, 2021.

[11] D. Casini, A. Biondi, and G. Buttazzo, "Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF," *IEEE Trans. Comput.*, vol. 70, no. 12, pp. 2168–2181, Dec. 2021.

[12] S. Liu, Z. Wang, G. Wei, and M. Li, "Distributed set-membership filtering for multirate systems under the round-robin scheduling over sensor networks," *IEEE Trans. Cybern.*, vol. 50, no. 5, pp. 1910–1920, May 2020.

[13] S. Kalapothas, G. Flamis, and P. Kitsos, "Efficient edge-AI application deployment for FPGAs," *Information*, vol. 13, no. 6, p. 279, 2022.

[14] L. Fick, S. Skrzyniarz, M. Parikh, M. B. Henry, and D. Fick, "Analog matrix processor for edge AI real-time video analytics," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, USA, Feb. 2022, pp. 260–262.

[15] D. Kanter. "Supercomputing 19: HPC meets machine learning." Website. 2019. https://www.realworldtech.com/sc19-hpc-meets-machine-learning/

[16] L. L. Zhang et al., "nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proc. 19th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2021, pp. 81–93.

[17] Z. Jia, J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken, "Optimizing DNN computation with relaxed graph substitutions," in *Proc. Mach. Learn. Syst.*, Stanford, CA, USA, Mar./Apr. 2019, pp. 1–13.

[18] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. 15th Eur. Conf. Comput. Vis.*, 2018, pp. 815–832.

[19] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent.*, New Orleans, LA, USA, May 2019, pp. 1–13.

[20] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.

[21] W. Liu, J. Geng, Z. Zhu, J. Cao, and Z. Lian, "Sniper: Cloud-edge collaborative inference scheduling with neural network similarity modeling," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jul. 2022, pp. 505–510.

[22] H. Lee, S. Lee, S. Chong, and S. J. Hwang, "Hardware-adaptive efficient latency prediction for NAS via meta-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, Dec. 2021, pp. 27016–27028.

[23] J. Zhao, K. Yang, X. Wei, Y. Ding, L. Hu, and G. Xu, "A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 305–316, Feb. 2016.

[24] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Feb. 2021.

[25] A. Shahroudnejad, "A survey on understanding, visualizations, and explanation of deep neural networks," 2021, *arXiv:2102.01792*.

[26] J. A. Alslie et al., "Áika: A distributed edge system for AI inference," *Big Data Cogn. Comput.*, vol. 6, no. 2, p. 68, 2022.

[27] S. Zhang, W. Li, C. Wang, Z. Tari, and A. Y. Zomaya, "DyBatch: Efficient batching and fair scheduling for deep learning inference on time-sharing devices," in *Proc. 20th IEEE/ACM Int. Symp. Clust., Cloud Internet Comput.*, May 2020, pp. 609–618.

[28] Y. Wang, G.-Y. Wei, and D. Brooks, "A systematic methodology for analysis of deep learning hardware and software platforms," in *Proc. Mach. Learn. Syst.*, Austin, TX, USA, Mar. 2020, pp. 1–14.

[29] L. Yang et al., "Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks," in *Proc. 57th ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jul. 2020, pp. 1–6.

[30] J. Adler and I. Parmryd, "Quantifying colocalization by correlation: The Pearson correlation coefficient is superior to the Mander's overlap coefficient," *Cytometry A*, vol. 77A, no. 8, pp. 733–742, 2010.

[31] A. Vaswani et al. "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30. Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[32] M. Dominic and B. N. Jain. "Conditions for on-line scheduling of hard real-time tasks on multiprocessors," *J. Parallel Distrib. Comput.*, vol. 55, no. 1, pp. 121–137, 1998.

[33] N. Mansouri, B. M. H. Zade, and M. M. Javidi, "Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory," *Comput. Ind. Eng.*, vol. 130, pp. 597–633, Apr. 2019.

[34] L. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: Prediction-based NAS using GCNs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10480–10490.

[35] Y. Choi, Y. Kim, and M. Rhu, "LazyBatching: An SLA-aware batching system for cloud machine learning inference," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 493–506.

[36] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving," in *Proc. USENIX Annu. Tech. Conf.*, Renton, WA, USA, Jul. 2019, pp. 1049–1062.

[37] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Boston, MA, USA, Aug. 2020, pp. 58–68.
[38] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in accelerating convolutional neural networks," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 2274–2283.
[39] M. Sbai, M. R. U. Saputra, N. Trigoni, and A. Markham, "Cut, distil and encode (CDE): Split cloud-edge deep inference," in *Proc. 18th Annu. IEEE Int. Conf. Sens., Commun., Netw.*, Rome, Italy, Jul. 2021, pp. 1–9.
[40] B. Zamirai, S. Latifi, P. Zamirai, and S. A. Mahlke. "SIEVE: Speculative inference on the edge with versatile exportation," in *Proc. 57th ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, 2020, pp. 1–6.
[41] R. G. Pacheco, R. S. Couto, and O. Simeone, "Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks," in *Proc. Int. Conf. Commun.*, Montreal, QC, Canada, Jun. 2021, pp. 1–6.

Weihong Liu received the B.S. degree from the School of Mathematics and Information Science, Guangzhou University, Guangzhou, China, in 2019. He is currently pursuing the Eng.D. degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.

His research interests include edge computing, deep learning, and task scheduling.

Jiawei Geng received the B.S. degree from the School of Management Science and Engineering, Dongbei University of Finance and Economics, Dalian, China, in 2020. He is currently pursuing the Eng.D. degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.

His research interests include edge computing, deep learning, and resource scheduling.

Zongwei Zhu received the M.S. and Ph.D. degrees in computer science from the University of Science and Technology of China (USTC), Hefei, China, in 2011 and 2014, respectively.

From 2014 to 2016, he was a Research Assistant with the IoT Perception Mine Research Center, China University of Mining and Technology, Xuzhou, China. From 2016 to 2018, he worked as a Senior Engineer with Huawei Company, Shenzhen, China. He is currently a Research Assistant with the Suzhou Institute for Advanced Research, USTC, Suzhou, China. His research focuses on resource scheduling, memory, power, and operating systems.

Yang Zhao received the Ph.D. degree from the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, China, in 2014.

He is a Senior Engineer with the Xi'an Institute of Space Radio Technology, Xi'an, China. His research interests include satellite network and satellite edge computing.

Cheng Ji received the M.E. degree from the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, in 2014, and the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, Hong Kong, in 2018.

He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include embedded systems, nonvolatile memory, and operating systems.

Changlong Li received the B.S. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2012 and 2018, respectively.

He was a Visiting Scholar with the University of California at Los Angeles, Los Angeles, CA, USA, and a Postdoctoral Fellow with the Department of Computer Science, City University of Hong Kong, Hong Kong. He is currently an Associate Professor with East China Normal University, Shanghai, China. His research interests include mobile devices, memory and storage systems, distributed systems, and IoT.

Zirui Lian received the B.S. degree from the School of Internet Finance and Information Engineering, Guangdong University of Finance, Guangzhou, China, in 2019. He is currently pursuing the Eng.D. degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.

His research interests include edge computing, swarm intelligence, and distributed machine learning.

Xuehai Zhou (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 1987, 1990, and 1997, respectively.

He is currently a Professor with the School of Computer Science, University of Science and Technology of China. He has lead many national 863 projects and NSFC projects. He has published more than 100 international journal and conference articles in the areas of software engineering, operating systems, and distributed computing systems.

Dr. Zhou serves as the General Secretary of Steering Committee of Computer College Fundamental Lessons, and Technical Committee of Open Systems, China Computer Federation.