ComHA: Cloud-Edge-Device Cooperative Model Building Based on Hierarchical Automated Machine Learning

Weiwei Lin, *Senior Member, IEEE*, Wangbo Shen, Wentai Wu, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Cloud-edge-device (CED) cooperative computing is an emerging paradigm that extends the reach of cloud services, providing higher flexibility and scalability to modern AI-driven computing services. However, traditional "one-size-fits-all" AI model construction at the edge struggles to accommodate the strong heterogeneity of target devices. This leads to an increasing demand for specialized models tailored for local resources in AI applications. To this end, we introduce ComHA, a cooperative model building framework based on hierarchical Automated Machine Learning (AutoML) to bridge the gap between hyperparameter optimization on the cloud and local model customization at the edge. In this framework, the cloud performs high-level AutoML to reduce the search space of learning algorithms, model architectures, and relevant hyperparameters to a specific set based on target device specifications. Subsequently, edge devices execute low-level AutoML to identify and train the optimal model, customized for their local data and resources. This approach aims to strike a balance between the benefit and cost of customized model building. Through extensive experiments conducted on a real-world testbed with public datasets, our results demonstrate that ComHA outperforms traditional methods in producing tailored models of high accuracy and low inference latency in various environments.**

***Index Terms*—AutoML, heterogeneous edge computing, machine learning, model customization.**

Received 20 November 2024; revised 24 October 2025; accepted 29 October 2025. Date of publication 3 November 2025; date of current version 13 July 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62072187 and Grant 62402198, in part by Guangdong Major Project of Basic and Applied Basic Research under Grant 2019B030302002, in part by Guangzhou Development Zone Science and Technology Project under Grant 2021GH10 and Grant 2023GH02, in part by the Basic Research Funds for the Central Universities under Grant 21624348, in part by the Major Key Project of PCL, China, under Grant PCL2023A09 and Grant PCL2025AS11, in part by Guangdong Provincial Natural Science Foundation Project 2025A1515010113, in part by Guangxi Key Research and Development Project under Grant 2024AB02018, in part by Shandong Provincial Natural Science Foundation Project ZR2024LZH012, and in part by the Basic and Applied Basic Research Foundation of Guangzhou under Grant 2025A04J2212. Recommended for acceptance by J. Castrillon. *(Corresponding author: Wentai Wu.)*

Weiwei Lin is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Pengcheng Laboratory, Shenzhen 518055, China (e-mail: linww@scut.edu.cn).

Wangbo Shen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: 202010107337@mail.scut.edu.cn).

Wentai Wu is with the Department of Computer Science, College of Information Science and Technology, Jinan University, Guangzhou 510632, China (e-mail: wentaiwu@jnu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TC.2025.3628026

I. Introduction

THE rapid development of 5G and Industrial Internet of Things (IIoT) technology motivates the edge-based application deployment where computation is closer to data. This also paves the way for AI-driven services that operate at the network edge. However, traditional cloud computing frameworks often fall short in delay-sensitive and heavy-traffic scenarios due to the long transmission distance between users and the cloud. By synergizing the computing capacity of the cloud and the heterogeneous resources at the edge, the CED collaborative computing offers a promising roadmap for the next-generation service provisioning. It is reckoned as a key enabler for AI workflows and has drawn broad attention from tech giants such as Google and Huawei due to its advantages in low latency, pervasive computing power, high capacity, and localization [1].

CED computing can facilitate collaboration across the layers of resource, application, and the capacity of intelligence. Beyond industrial practices, the CED framework has proved effective in diverse domains such as intelligent transportation systems [2], healthcare systems [3], and smart home applications [4], highlighting its adaptability across various contexts.

Although the CED collaboration framework has inherent advantages AI service deployment, the strong heterogeneity of edge and end devices remains a major challenge. As shown in Table I, if models are not adapted across heterogeneous devices, it can significantly compromise inference latency, memory usage, and power consumption. For instance, on the Atlas 200 DK, inference latency without adaptation is as high as 53.18 seconds, which improves dramatically to 13.41 seconds after adaptation. As a result, it is a great challenge to jointly utilize the resources on the cloud and those at the edge to produce models that fit perfectly on specific devices [5]. Due to the existence of a large number of heterogeneous devices, cloud-trained AI models need to be adapted before deployment at the edge. As heterogeneous devices and AI chips flood the market, model adaptation brings heavy burden tothe AI service providers.

TABLE I
PERFORMANCE COMPARISON BETWEEN ADAPTED (LOCALIZED) MODELS AND MODELS DELIVERED DIRECTLY FROM THE CLOUD. SHORTHANDS: IT=AVERAGE INFERENCE LATENCY, OM=AVERAGE OCCUPIED MEMORY, RU%=AVERAGE RESOURCE UTILIZATION

Running w./w.o. adaptation	IT/s	OM/mb	RU/%	Power/W
Cloud server	1.21	4055.13	24.75	9-17
Atlas 200 DK	53.18	1273.15	15.53	0-0.1
Atlas 200 DK (**adapted**)	13.41	945.05	11.53	0-0.1
PC	7.75	909.76	11.10	5-10
PC (**adapted**)	1.88	2376.63	29.01	3-6
Raspberry PI	49.55	1303.37	31.81	0-0.1

AutoML, as an increasingly prevalent AI technology in recent years, provides a fast way to model construction through automatically configuring model structures and optimizing training hyperparameters [6], [7], [8]. It offers a native approach to finding the optimal model, tailored to their specific local architectures and datasets. However, the computational demands of AutoML can be prohibitive regarding the standalone capabilities of edge devices to execute the entire process autonomously. Within the realm of cooperative model building, the challenges posed by AutoML include:

- **Models by cloud-centric AutoML methods struggle at the edge device**: Existing AutoML approaches predominantly operate at the cloud level and are not designed to generate specialized model designs adaptable to the diverse hardware architectures and resource constraints found at the edge. This mismatch leads to sub-optimal model performance when deployed on heterogeneous edge devices.
- **Resource constraints hindering independent AutoML at the edge**: Edge and end devices often lack the computational capacity and energy resources to independently conduct AutoML processes. The high computational demands make edge-based AutoML impractical or cost-prohibitive on resource-constrained devices.
- **High latency and network overheads in cloud-based solutions**: Relying solely on cloud-based model training and optimization necessitates transmission the model itself, resulting in significant latency and network overheads. In latency-sensitive edge scenarios, such as real-time monitoring and control systems, cloud-centric methods struggle to meet the stringent real-time requirements.

Neither solely cloud-based nor edge-based AutoML approaches can independently address the challenges that exist between the cloud and the edge. To this end, our proposed framework, ComHA, strategically utilizes hierarchical AutoML to tackle these inherent disparities and resource constraints. ComHA avoids "one-size-fits-all" AI model construction and is designed as a CED AutoML framework with the primary objective of accelerating local model personalization via architectural search. This means that ComHA saves a lot of effort for users who desire a device-customized local AI model but does not afford full-space AutoML. ComHA aims to find the best-suited local model and does not exclude the deployment of a strong cloud-based model. In this case the user can route their request flexibly to enjoy either fast local inference or high-quality remote reasoning.

By leveraging hierarchical AutoML, ComHA facilitates a seamless integration of model architecture and hyperparameter search and localized model deployment. Initially, the cloud undertakes high-level AutoML to confine the search space of learning algorithms, model architectures, and hyperparameters to a specific subset. Subsequently, edge devices engage in low-level AutoML activities to search for and train the optimal model tailored to their local data and resource constraints. This novel approach harmonizes the benefits and costs associated with custom model building, addressing the challenges presented by device heterogeneity. It also avoids any incompatibility in model conversion by means of native training.

Our comprehensive experiments conducted on a real-world testbed plus a case study employing public datasets underscore the effectiveness of ComHA in automating model construction across heterogeneous devices. Notably, our results showcase ComHA's superiority over existing methodologies, particularly in enhancing model accuracy and reducing inference latency. In summary, we make the following key contributions:

1) We designed a cooperative model building framework based on hierarchical AutoML that significantly eases the burden of AI model design and adaptation.
2) Our empirical finding shows that existing model evaluation criteria are insufficient to reflect the actual performance of AI models running on heterogeneous devices, which indicates the importance of local model adaptation.
3) Through extensive experiments on a real-world testbed and a case study, we demonstrate that the models trained using our ComHA framework can adapt well to various heterogeneous edge devices with superior performance over existing approaches.

The rest of this paper is organized as follows. We introduce the research background and summarize related studies in Section II. In Section III and IV we outline our ComHA framework and detail the algorithm design. Section V presents our evaluation results followed by deeper analysis in Section VI. In Section VII we conclude our work.

II. BACKGROUND AND RELATED WORK

A. Cloud-Edge-Device Computing Framework

Different from traditional cloud computing architecture, cloud-edge-device collaboration provides faster and better computing services for end devices by sinking part of the computing resources closer to the end devices [5]. Furthermore, the cloud serves as the coordinator in the architecture providing overall management, as shown in Fig. 1.

How to cooperate effectively and efficiently is the most critical question for a cloud-edge-device collaborative computing system [1]. Laskaridis et al. [9] proposed a cloud edge collaborative reasoning model named SPINN, which places multiple early exits in the neural network. When the confidence of the neural network is good enough, the inference will be stopped, and the current results will be output. This method can be deployed in the edge-cloud cooperative system, which

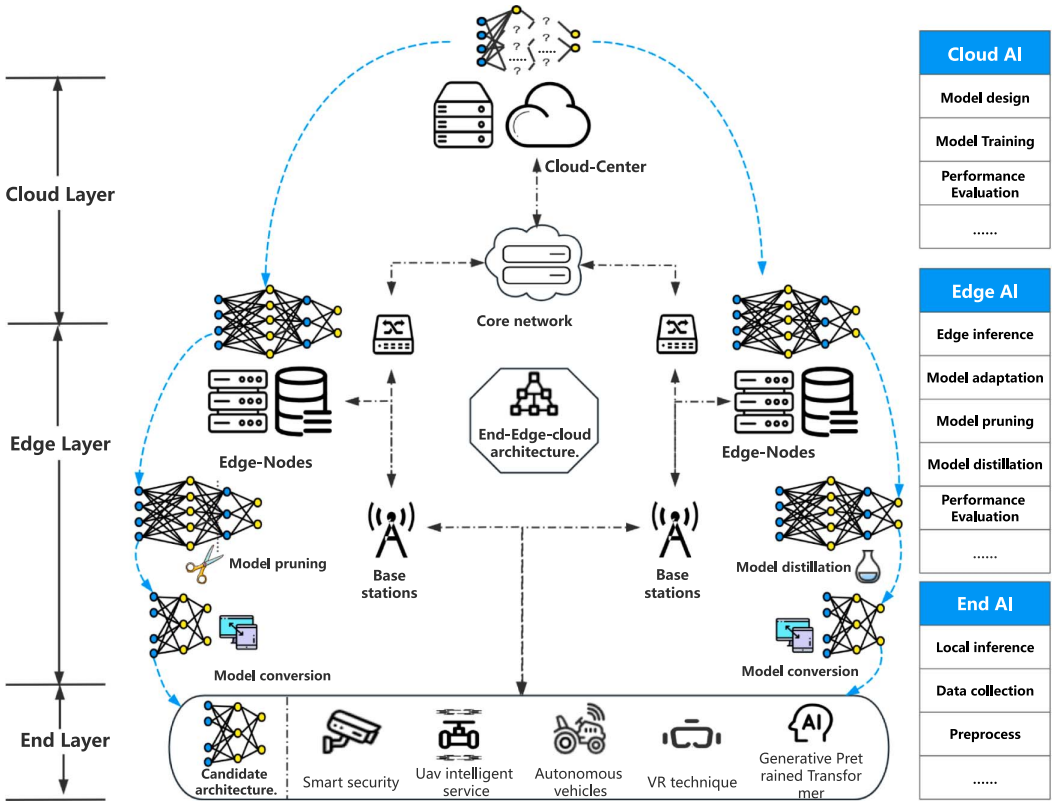

Fig. 1. In the traditional cloud-edge-device computing framework, AI processes are segmented across end, edge, and cloud devices, aiming to enhance service quality and speed by utilizing collaborative technologies. However, this framework predominantly emphasizes model training and inference performance, overlooking the complexities of model adaptation on diverse devices. This oversight hampers the seamless deployment of models across the expanding landscape of heterogeneous devices flooding the market.

considers the shortage of edge-side resources, the instability of the cloud, and the cost of communication. In [10], the authors presented a dynamic DNN decomposition system and introduced a heuristic algorithm for collaborative inference without the loss of accuracy. Wu et al. [11] put forward a personalized federated learning framework for intelligent IoT applications in a cloud-edge collaborative computing architecture, which can reduce the negative impact of heterogeneity in different aspects. Ref.[12] offered an incremental learning method for abnormal object detection by combining cloud-edge-device collaborative computing-integrated collaborative computing architecture with deep learning technology.

A number of platforms and systems have been developed following the cloud-edge-device collaborative computing architecture in the industry to facilitate applications such as water-cooling control [13], satellite ground stations [14], and short-term power load forecasting [15]. In [16], a temperature measurement system for COVID-19 was built based on the cloud-edge-device collaborative architecture. In ref.[17], the authors improved the privacy protection of the food recommendation system through cloud-edge collaboration.

B. Automated Machine Learning

AutoML is a novel technology that automates the design of model structures and the adjustment of hyperparameter settings. As an important recipe for deep neural network customization, AutoML has gained much attention from both the academia and the industry. A diversity of approaches have been developed in the field of AutoML including Neural Architecture Search (NAS) [18], Hyperparameter Optimization (HPO) [19], Combined Algorithm Selection and Hyperparameter Optimization (CASH) [20], Automated Data Mining (ADM) [21], Automated Reinforcement Learning (AutoRL) [22], Meta-Learning and Learning to Learn (Meta-L) [23], Bayesian Optimization for AutoML (BOA), Evolutionary Algorithm for AutoML(EA) [24], Multi-Objective Optimization for AutoML(MOO) [25], AutoAI [26].

By automating model design and fine-tuning, AutoML plays a pivotal role in tailoring models precisely to the specific constraints and requirements posed by edge devices, such as limited computational resources and stringent latency constraints. This adaptability is crucial for optimizing model construction and adaptation processes in the context of diverse edge devices.

C. Hardware-Aware Model Performance Evaluation

The primary goal of collaborative model building is to produce a model that well adapts to a specific edge device or end device in terms of accuracy, inference latency, and energy consumption, etc. Therefore, models built and trained on the cloud must be evaluated in the above metrics before sending them to the network edge for running on different devices.

At present, the most commonly-used metrics for evaluation are model structure-related, such as FLOPs, the number of parameters, and the number of layers. However, they are not

hardware-aware and thus cannot accurately reflect the actual performance of AI models on different devices [27]. A naive solution is to establish a lookup table for any particular devices, but this is very time-consuming and costly. For example, in [28], the authors measured the average inference latency of 5,000 sample DNNs and used the results to construct a delay lookup table for that particular device. However, Lu et al. [29] figured out that it may take more than 27 hours to build a latency predictor for a single device for each kind of operator. To this end, they opted for a proxy adaptation technique to significantly boost the latency monotonicity so that a Pareto-optimal architecture can be found almost identical to the existing single-device NAS using only one proxy device. However, building such a predictor still consumes a lot of time and resources, considering the rapid update of devices and computing architecture in the current market.

The main reason why existing AutoML methods fall short is that there are so many heterogeneous devices that it is almost impossible to standardize and unify the adaptation. Statistics showed that there was no typical system-on-chips (SoC) in the market [30]. The most commonly used SoCs account for less than 4 percent of the total market. As a result, it is very challenging to evaluate an AI model trained on the cloud regarding its performance on a particular edge device or end device. Hence we argue that instead of relying on prediction of model performance after training on the cloud and then adapting the model to heterogeneous devices, an alternative solution is to leverage the cloud for automated model design and perform the training locally on the target device.

It is worth mentioning that we also found from our experiments that existing model evaluation criteria are insufficient to reflect the actual performance of AI models running on heterogeneous devices, indicating the importance of local model adaptation. The corresponding experimental results and content are presented in Section V.

III. Overview of the Proposed Framework

The major difference between our ComHA framework and the AI model development process in classical cooperative architecture is that the cloud in ComHA does not simply send a trained AI model to the edge or end devices. Instead, it sends the optimal AutoML configurations to the target device and skips the adaptation stage. Through subsequent local AutoML, one can quickly train the model that is most suitable for the local environment and data without any adaptation. Fig. 2 visualizes the comparison.

This section introduces the process of model building in ComHA which is a two-stage procedure sequentially executed by the cloud server and the target edge or end device, comprising a hierarchical AutoML pipeline.

A. ComHA Stage I – Cloud-Level AutoML

The architectural design and search for model hyperparameters are very time-consuming. By design the edge and end devices should minimize the burden in this regard due to limited resources. With its colossal computing resources, the cloud is the best option for the majority of AutoML workload.

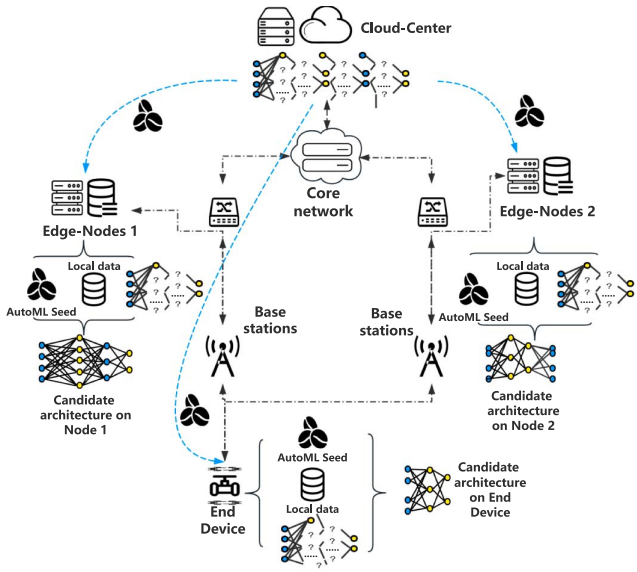

Fig. 2. ComHA replaces explicit model conversion with hierarchical AutoML, which is a completely different pipeline than traditional proxy-based approaches.

In the collaborative process of ComHA, the cloud is mainly responsible for selecting a small set of candidate model architectures through AutoML in a vast search space. Apart from the type of model and its structural parameters, the cloud also needs to define the corresponding parameters for the following edge-level AutoML.

The cloud first attempts to instantiate an instance of the same architecture as the target edge/end device, such as GPU and Tensorflow, Ascend-310 and ACL framework, etc. In this way it avoids the adaptation of models on edge/end devices. Also, it can more accurately predict the performance of cloud-trained models running on edge/end devices. In the case when the cloud does not have homogeneous computing resources or architecture available, the most efficient training architecture is used.

After selecting the chip and architecture, the cloud defines the evaluation metrics and constraints for the model. The set of metrics includes model accuracy, inference latency, energy consumption, memory footprint, etc. The cloud also needs to set multiple training constraints to prevent potential underfitting or overfitting.

The stage of cloud-level AutoML aims to find the optimal model topology and hyperparameters by traversing various AutoML strategies such as HPO, NAS, EA, and model classes such as DNN, CNN, RNN. The selection of models is not limited to neural networks, classical machine learning algorithms such as SVM and Random forest are also considered [31]. Once this is done, the cloud packages them into a triplet and send them to the edge/end device, as shown in the Fig. 3.

Generally speaking, a vast search space will be set at the beginning of this training process to ensure that the training results of the AutoML system can be infinitely close to the global optimal. By contrast, the resulting triplet corresponds to

Fig. 3. The cloud searches for the best combination (namely, a "seed") of model architecture, AutoML method, and their hyperparameters.

a much smaller search space, which offers flexibility for the target device to find the optimal model variant without intensive search and training.

B. ComHA Stage II – Edge-Level AutoML

The conventional approach to deploying models on edge and end devices is to adapt pre-trained models directly. However, as hardware platforms and AI algorithms evolve rapidly, adaptation tools struggle to keep compatible with new architectures and API libraries. For example, converting Huawei Ascend's model native format with the ATC tool applies default FP32 to FP16/INT8 quantization, which yields high error when the calibration data are insufficient. Unsupported ONNX operators will also be approximated in ways that compromise both inference speed and numerical precision. More importantly, model conversion often fails to leverage local compute resources efficiently, resulting in compromised performance.

As shown in Fig. 2, in the ComHA framework, after the model architecture search and parameter optimization are completed in the cloud, the edge/end device only needs to input the "seed" from the cloud into the local AutoML system to generate the AI model that is most suitable for the local computing environment. In this process, the optimization of topology and hyperparameters is performed in a relatively small solution space using a single AutoML algorithm (e.g., NAS or CASH) designated by the cloud. This greatly reduces the workload on the device whilst allowing for local exploration based on local data and evaluation in local computing environment. In other words, the edge-level AutoML system in ComHA selects the AI model most suitable for the local environment based on the actual model performance data collected locally. This is by design a more native way to produce device-friendly models than existing cloud-based model design approaches.

It is worth noting that we enable the stage II of ComHA by default but it is up to the users to decide if it is necessary. In practice, the local AutoML stage is encouraged for training-capable devices such as edge servers with GPUs or training-oriented NPUs/accelerator cards. For inference-oriented devices (e.g., platforms with Atlas 200DK NPU), one can skip the stage and optionally resort to local adaptation based on fine-tuning on the local dataset as employed in [32].

IV. PROBLEM FORMULATION AND ALGORITHM DESIGN

A. Problem Description

In this paper we consider a problem setting as the following. The cloud server(s) in the system has a certain volume of data, denoted by D, for training. Also, there are n different classes of machine learning models available for the task. By $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ we denote this set of model classes, where M_i represents a machine learning model. We assume that there are m kinds of AutoML algorithms applicable in the Cloud-edge-device computing framework, denoted as $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$, where A_i represents an AutoML algorithm. An AutoML algorithm A_i typically needs to define a specific set of hyperparameters to match a given class of model M_i. Here, we use e_{ij} to denote the hyperparameters of the j-th AutoML algorithm for the i-th machine learning model, and by $\xi = \{e_{11}, e_{12}, \ldots, e_{ij}, \ldots, e_{nm}\}$ we denote the set of AutoML hyperparameters. With this definition, the search space of ComHA is $> \mathcal{M}, \mathcal{A}, \xi <$ where we call each feasible solution a "seed" because it will be sent to the target device as the search setting for edge-level AutoML (Fig. 3).

The objective of ComHA, at cloud level, can then be described as finding the optimal seed $B =< M_B, A_B, e_B >$ for AutoML, where B and resulting model θ_B need to satisfy the following conditions:

$$M_B \in \mathcal{M}, A_B \in \mathcal{A}, e_B \in \xi, \tag{1}$$

$$\text{Delay}(\theta_B) \leq \text{Time}_D, \tag{2}$$

$$\text{Acc}(\theta_B) \geq \text{Accuracy}_D, \tag{3}$$

$$E(B) \leq \text{Energy}_D, \tag{4}$$

$$\text{Mem}(B) \leq \text{Memory}_D, \tag{5}$$

$$\text{AutoML}(B)\ s.t.\ \text{Constraints}, \tag{6}$$

where $\text{Delay}(\theta_B)$ and $\text{Acc}(\theta_B)$ represent the inference delay and accuracy of the resulting model, $E(B)$, and $\text{Mem}(B)$ stand for the energy consumption, and memory footprint for the AutoML process given B, respectively. Time_D, Accuracy_D, Energy_D, and $Memory_D$ represent users' requirements on the inference delay, accuracy rate, energy consumption, and memory usage for analyzing data D; $\text{AutoML}(B)$ represents the process of automated learning with B and Constraints represents the constraints for model evolution.

B. Algorithm Design of ComHA

The key factors in the algorithm design of ComHA are the hyperparameters setting of the AutoML algorithm and the definition of evaluation metrics.

In order to better explain the concept of AutoML hyperparameters, we take NAS as an example and follow the notations in [33]. There are two categories of hyperparameters for NAS: 1) the $Pipeline$, which defines the model structure-related hyperparameters such as the dimensions of layers; 2) the $RandomSeed$, which determines how the model is initialized. For ease of presentation, by P and R we denote the hyperparameter set of $Pipeline$ and $RandomSeed$, respectively. For deep neural networks (DNNs), the basic network structure

consists of an input layer, one or multiple hidden layers, and an output layer, where the input layer and output layer are fixed and determined by the task [34]. In other words, the parameters that need to be automatically optimized by NAS are mainly the dimensions of the hidden layers. In this case, the hyperparameter setting e can be written as $e =< P, R >$, where P represents the DNN's hidden layer search space and R represents the parameter initialization method.

Evaluation metrics for AutoML essentially serve as the criteria to gauge a trained model. As mentioned in Section III, typical metrics are model accuracy, inference latency, energy consumption, memory occupation, etc. Since AI services can have complex requirements such as fast response, quality reasoning and low power consumption, we simplify the multi-objective optimization using a composite score for candidate solution evaluation. The score is a weighted sum of the traditional metrics:

$$\begin{aligned}\text{score}(B,\theta_B) = \alpha\text{Acc}(\theta_B) + \beta\text{Delay}(\theta_B) + \\ \gamma E(B) + \delta\text{Mem}(B),\end{aligned} \tag{7}$$

where

$$\alpha + \beta + \gamma + \delta = 1.0, \tag{8}$$

$\text{Acc}(\theta_B), \text{Delay}(\theta_B), E(B), \text{Mem}(B)$ represent the best model accuracy, inference delay, energy consumption, and required memory by an AutoML solution B measured on the server. α, β, γ, and δ are the corresponding weights depending on the preference of users and the requirements of the task. For example, one can set a large β for delay-sensitive tasks and a small β for delay-tolerant tasks [35].

Our scoring method can be easily extended to different AutoML algorithms where more factors such as loss functions and evolutionary direction of the population can be considered. The overall model building process of ComHA is shown in Algorithm 1.

In Algorithm 1, the $\text{autoML}(M_j, A_i, e_{ij}, D_C)$ function performs cloud-level automated training using a specified AutoML algorithm A_i and returns the corresponding seed B, the trained model θ_B as well as the model type κ_B (e.g., the number of hidden layers and the initialization method for the model). The best seed encapsulates an expanded model class κ_B^+, the search algorithm A_i and the corresponding hyperparameters e_{ij}. On the target device, AutoML at the edge-level will be based on all these configurations specified in the best seed B^+ and the local data D_E, ultimately returning the refined parameters θ_B^* and hyperparameters κ_B. Note that AutoML is performed both in the cloud and on the target device, but the search space for the latter is significantly reduced with the choice of algorithm fixed.

In order to avoid falling into local optimum, edge/end devices further explore the vicinity of the search space defined by the κ_B, which means they search for a potentially better network structure by using κ_B as the initial point and expanding it a little bit as the search space for edge-level AutoML training. This is the function of $\text{expand}()$. For example, if κ_B defines four hidden layers, and the maximum number of neurons in the hidden layers is 12, then the hidden layer search space of edge-level AutoML is [3, 4, 5], and the maximum dimension range is [1, 13].

Algorithm 1: The hierarchical AutoML in ComHA

Data: Data on the cloud D_C; On-device data D_E; Machine learning model classes $\mathcal{M} = \{M_1, M_2, ..., M_n\}$; AutoML algorithms $\mathcal{A} = \{A_1, A_2, ..., A_m\}$; AutoML Hyperparameters $\xi = \{e_{11}, ..., e_{nm}\}$; Constraints for cloud-level AutoML to continue; the scoring function $score()$ for evaluation.

Result: Optimal AI model θ^* and hyperparameters κ^*

$B^* = null, \kappa_B^* = null, \theta_B^* = null$;

`//* Cloud-level AutoML *//`;

while Constraints *are still satisfied for the process* **do**

  for *i=0 to m* **do**

    for *j=0 to n* **do**

      $\theta_B, \kappa_B, B = \text{autoML}(M_j, A_i, e_{ij}, D_C)$;

      if $\text{score}(B, \theta_B) \geq \text{score}(B^*, \theta_B^*)$ **then**

        $B^* = B$;

        $\theta_B^* = \theta_B$;

        $\kappa_B^+ = \text{expand}(\kappa_B)$;

        $B^+ =< \kappa_B^+, A_i, e_{ij} >$;

The cloud sends B^+ to the target device;

`//* target device-level AutoML *//`;

$\theta_B, \kappa_B = \text{autoML}(B^+, D_E)$;

$\theta* = \theta_B$;

$\kappa^* = \kappa_B$;

return κ_B, θ_B^*

V. Experiments

A. Datasets

In this section, we evaluate the proposed framework ComHA on three datasets on real-world testbed.

CIFAR-10 is composed of 32 × 32 RGB images belonging to 10 different classes. The dataset is partitioned into two sets: a training set with 50,000 images and a test set with 10,000 images.

CIFAR-100 consists of 60,000 color images of size 32×32 pixels, divided into 100 fine-grained classes with 600 images each (500 for training and 100 for testing). These 100 classes are further organized into 20 superclasses, and the dataset is partitioned into 50,000 training images and 10,000 test images.

MNIST is a popular dataset for handwritten digit recognition with 10 classes (0-9). The training set contains 60,000 images and the test set contains 10,000 images. All samples are 28×28 grayscale images.

FASHION-MNIST is composed of 28×28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 images and the test set has 10,000 images.

B. Environment Setup

The experiment environment consists of heterogeneous end devices, edge servers, and a cloud server as the coordinator. Their configuration is as follows:

Cloud server: 1 X86 server equipped with 16 Intel (R) Xeon (R) Gold 5218 @2.30GHz with 2 Nvida Tesla T4 GPU (16G);

Edge server: 3 Atlas 200 DK equipped with 2 A55 Arm core@1.6GHz with one Davinci AI core NPU (8G); and one PC (MECHREVO notebook) with Intel(R) Core(TM) i7-6700HQ CPU @2.60GHz with a Nvidia Geforce GTX 1060 GPU (8G).

End device: 4 Raspberry Pi equipped with 4 ARM cortex-A72@1.5GHz with Broadcom VideoCore VI@500MHz (4G).

In terms of AutoML algorithms for our framework, this paper chooses the NAS method of Random Search and Evolutionary Algorithms [24] to conduct experiments. Our framework can be easily extended to many more AutoML technologies in [36].

C. Evaluation Results

In this section, we demonstrate experimental results that answer the following questions: 1) *Can existing methods and indicators accurately reflect the actual performance of models to be run on edge/end devices?* 2) *How much better is a locally trained or adapted model than that without adaptation?* 3) *Can the ComHA framework effectively produce high-performance models for heterogeneous devices?*

1) Existing Model Performance Indicators Versus Actual Latency: **AutoML algorithm.** In order to verify the performance of the AI models on different devices, we used the NAS method of Random Search on the cloud server and randomly trained 2000 CNN models with Tensorflow2.0 on the MNIST dataset. The search space settings of CNN are as follows: The depth of the model is "`Layer = 1,2,3,4,5,6,7,8,9`", the size of the convolution kernel of each layer is "`kernel = 1,2,3,4,5(CONV2D)`", the size of the filter of each layer is "`filter = 2,4,8,16,32,64,128,256,512`", and the initialization method of model parameters is "init=`Xavier, Kaiming, normal distribution`.", "`batch size=[64,128,512], epoch=`[5], [12], [20]".

Model performance indicators. At present, there are four widely used indicators to measure the complexity of a model: FLOPs, the number of parameters, MACs, and the number of layers. Considering the fact that MACs has a linear relation with FLOPs, we in this paper use FLOPs to represent both FLOPs and MACs. More information about these indicator can be found in [37].

Actual performance measure. To measure the actual performance of models on the target device, this paper primarily uses model inference latency as the performance indicator, measured using a test set sample entered at $batch\_size$=1. Since the model structure and parameters remain unchanged during the migration process across different computing architectures or chips, the inference accuracy of the model remains unaffected. However, the actual inference latency of the model is significantly impacted by the heterogeneous computing architectures and chips, particularly within the current cloud-edge-device computing framework. As a result, inference latency often becomes the most critical concern for users.

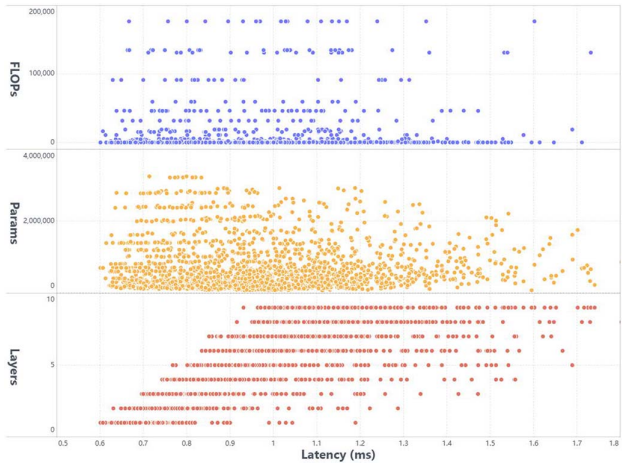

Fig. 4. The correlation between FLOPs, Params, Layers, and actual inference latency.

Then, we collected and compared the performance of the 2,000 CNN models to examine the correlation between actual inference latency and the performance indicators (FLOPs, params., and number of layers), which is visualized in Fig. 4.

2) Importance of Model Adaptation or Localization: **Device.** In this part of experiment, one Atlas 200 DK, one PC, and one Raspberry PI are selected to deploy the computing model delivered by the cloud.

Model. For ease of presentation, we select a subset of 100 models out of the 2000 CNN models automatically trained by NAS technology on the cloud, as mentioned in the previous section V-C1. For deployment at the edge/end, only a subset of the 2000 computing models is utilized. Another reason for not testing all the models comes down to the fact that most of the models, without proper adaptation, yield extremely poor performance on the target device. Hence, we choose to demonstrate the result with only the first 100 models in this scenario.

Model performance indicators. In this section, inference latency, occupied memory, resource utilization, and energy consumption are selected as the four indicators to measure the performance of the calculation model during operation.

For power evaluation on the cloud server, the IDRAC method is employed for periodic power acquisition. The power consumption of the PC is measured with a power management software. As for Raspberry PI and Atlas 200 DK, their power consumption is measured using built-in energy consumption query instructions.

The remaining three metrics (inference latency, occupied memory, resource utilization) are recorded as measured data from the device running the model and the memory-profiler library.

Method of adaptation. The available devices for adaptation are the PC and the Atlas 200 DK. The cloud computing model is trained using Tensorflow 2.10.0, while the PC environment uses different versions of Tensorflow (2.6.0), CUDA and CUDNN.

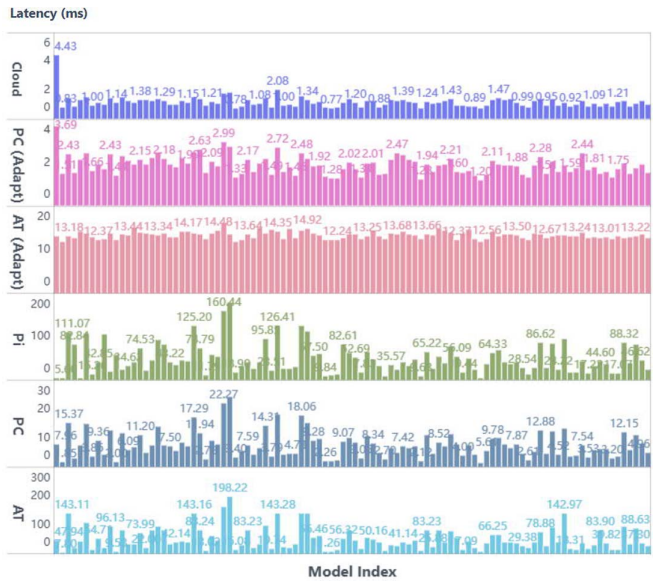

Fig. 5. Inference latency of the 100 CNN models run on different devices. Adapted or localized models yield significantly lower latencies.

TABLE II
COMHA PARAMETER SETTINGS ON THE THREE DATASETS

Dataset	Population	Evolution	Fit params	
			Batch size	Epoch
MNIST	5	500	512	3
FASHION-MNIST	5	500	512	5
CIFAR-10	5	100	64	12

To run the cloud-delivered model on the PC, there are two options. The first option is to update TensorFlow to version 2.10.0 and run the model using CPU. The second option is to update TensorFlow to version 2.10.0 and also update CUDA and cudnn to versions 11.2 and 8.7 respectively for GPU adaptation and execution (adapted).

Atlas 200 DK is equipped with NPU and tensorflow 2.10.0, so there are two ways to run the cloud-delivered model on the Atlas 200 DK. One is to run the model on CPU directly (no adaptation). The second way is to first convert the model built on the cloud (in.h5 format) to the device-specific format (.pb) for Atlas and then finish adaptation via the Ascend Tensor Compiler (ATC) tool for Atlas NPU (adapted).

We design two sets of experiments, respectively, on the devices with heterogeneous computing architectures (Mechanical MECHREVO with GPU and Atlas 200 DK with NPU) to draw comparison between the performance of models adapted locally and those directly from the cloud without adaptation. Fig. 5 shows the inference latency of these 100 models.

3) Performance Evaluation on Edge/End Devices: **Device.** In this part of experiment, one Atlas 200 DK, one PC are selected to deploy the models delivered by the cloud.

Dataset. This section selects MNIST, FASHION MNIST, and CIFAR-10 as experimental data.

ComHA parameter setting. For demonstration we select Evolutionary Algorithm (EA) as the AutoML algorithm and show the parameter settings of ComHA in Table II, where Population and Evolution represent the population size and the maximum number of generation in the evolution. The scoring weight parameters α, β, γ, and δ for model selection are 0.5, 0.3, 0.1, and 0.1, respectively. These values are set based on the mindset that model performance usually matters the most. This makes sense in many industrial scenarios where accuracy and inference latency are prioritized over energy consumption and memory footprint. In practice, these parameters can be tweaked by demand.

TABLE III
COMPARING THE MODEL ADAPTATION AND AUTOML METHODS IN OUR EVALUATION

	DLMC/ATC	OFA	ComHA
AutoML-based		✓	✓
Local training-free	✓	✓	
Model adaptation-free			✓
Local resource-aware		✓	✓

Model performance indicators. In this section, accuracy, inference latency, occupied memory, and power are selected as the four indicators to measure the performance of the model during operation.

Baseline methods. ComHA represents a hierarchical, collaborative model building approach, integrating cloud, edge, and end devices. In this section, we primarily compare it with the conventional practice where the model is trained on the cloud followed by adaptation at the edge for deployment. The key technique for these baseline methods is the Deep Learning Model Converter (DLMC),[1] which is a generic model adaptation tool for GPUs, and ATC, which is developed by Huawei for NPUs [38]. In our experiments, we simulate DLMC's workflow by using MMdnn to export a TensorFlow model to ONNX before performing the adaptation step.

Additionally, we compare ComHA with the state-of-the-art deployment framework Once-For-All (OFA) [39], which adopts an adaptation-based strategy without local retraining. OFA performs automated architecture search on a large supernet in the cloud to identify compact sub-networks that trade a slight accuracy reduction for significantly smaller model size, making them well suited for edge and end devices. Table III summarizes their key differences.

Note that we skip the progressive shrinking in OFA but focus instead on the final model's adaptability across heterogeneous devices. Given the simplicity of the model and dataset in this section, we used an exhaustive training method. The search method for edge nodes adheres to the same EA as the original paper, with model selection and training guided by accuracy and MACs (i.e., Computational Resource Constraints: PC 100M, Atlas 33M) indices.

In Table IV and Fig. 6, we compare the performance of the ComHA and cutting-edge model adaptation methods on the PC side (GPU) and Atlas 200 DK (NPU). All the latency and memory reported in Table IV are the average results of ten runs. The structure of models is expressed compactly as "(the number of hidden layers, size of convolution kernel, and size of filter)". For the discrepancies in memory footprint across devices, it

[1] https://github.com/ysh329/deep-learning-model-convertor

TABLE IV
COMPREHENSIVE PERFORMANCE COMPARISON BETWEEN COMHA AND THE BASELINE MODEL BUILDING METHODS

Device	Dataset	Method/tools	ACC	Latency (s)	Memory (MB)	Power (W)	Structure
PC	MNIST	ComHA	99.18	**1.15**	**2095**	5	(5,[4, 2, 4, 1, 4],[128, 128, 64, 64, 64])
		DMLC	99.18	1.32	2124	5	(5,[4, 2, 4, 1, 4],[128, 128, 64, 64, 64])
		OFA	99.18	1.17	2380	5	(5,[4, 4, 4, 1, 2],[64, 16, 128, 16, 16])
	FASHION-MNIST	ComHA	**91.44**	**0.65**	2003	5	(3,[2, 2, 3],[32, 32, 128])
		DMLC	**91.44**	1.24	2016	5	(4,[3, 1, 2, 4],[32, 32, 128, 128])
		OFA	89.74	0.77	**1850**	5	(3,[2, 2, 3],[32, 32, 128])
	CIFAR-10	ComHA	**79.10**	**2.35**	**3812**	7	(3,[2, 2, 2],[128, 128, 512])
		DMLC	77.43	3.14	4534	7	(3,[2, 4, 1],[256, 256, 256])
		OFA	77.43	3.10	4478	7	(3,[2, 4, 1],[256, 256, 256])
Atlas	MNIST	ComHA	**99.18**	11.99	897	0.1	(5,[4, 2, 4, 1, 4],[128, 128, 64, 64, 64])
		ATC	**99.18**	13.85	956	0.1	(5,[4, 2, 4, 1, 4],[128, 128, 64, 64, 64])
		OFA	98.83	**10.36**	**816**	0.1	(3,[2, 2, 4] [128, 128, 128])
	FASHION-MNIST	ComHA	**91.44**	13.48	699	0.1	(4,[3, 1, 2, 4],[32, 32, 128, 128])
		ATC	**91.44**	13.56	689	0.1	(4,[3, 1, 2, 4],[32, 32, 128, 128])
		OFA	88.95	**12.15**	**612**	0.1	(3,[1, 2, 2],[64, 128, 128])
	CIFAR-10	ComHA	**79.65**	**13.56**	1252	0.1	(3,[2, 4, 1],[128, 256, 512])
		ATC	77.43	15.81	1289	0.1	(3,[2, 4, 1],[256, 256, 256])
		OFA	74.26	14.35	**1092**	0.1	(3,[2, 2, 2],[32, 128, 512])

mainly boils down to the runtime environment. GPU platforms load the full CUDA runtime and its libraries when initializing an AI model, which substantially inflates memory usage. In contrast, energy-constrained devices such as the Atlas200DK and Raspberry Pi preload only a minimal set of dependencies, resulting in much smaller footprints. On the CIFAR-10 dataset, ComHA occupies less memory and uses shorter latency. This is because ComHA produces the final localized model through local training whilst ATC is an adaptation-based method that converts a remotely-trained model to fit the local hardware.

Stage-wise training cost. To reveal why it is important to reduce search space at the edge/end device, we first report the per-iteration model training cost for different devices: For MNIST and FASHION-MNIST (with a batch size of 512), the average wall-clock time per iteration is approximately 3s∼4s on the PC, 280s∼300s on the Atlas200DK, and 0.6s∼1.1s in the cloud. For CIFAR-10, each model was trained for 12 epochs with a batch size of 64, requiring about 840s per iteration on the PC, 430 minutes on the Atlas200DK,[2] and 98s in the cloud.

The huge discrepancy in training speed validates the fact that reducing the search steps on edge/end devices is crucial for the collaborative model building process. Fig. 7 tracks the number of iterations in each stage for ComHA on the three data sets. It is evident that ComHa effectively reduces the number of iterations at the edge (i.e., PC and Atlas, stage II). This is realized by having the cloud, which is at least two magnitudes faster than Atlas, iterate for significantly more steps (stage I) so as to refine the search space as much as possible. As a result, edge devices are able to find a strong solution within much fewer steps (especially on CIFAR-10), leading to faster model deployment. Although the per-iteration wall-clock times vary substantially across the evaluated devices, it does not undermine the feasibility of our solution. ComHA reduces the number of training iterations—and hence the time-to-accuracy—for any device under its own resource budget. In practice, one can skip the stage II and resort to fine-tuning on weak devices if they are not training-friendly.

[2]Although the Atlas200DK is equipped with an NPU for accelerated inference, it does not support on-chip training and therefore relies on its onboard CPU for training, resulting in performance comparable to that of a Raspberry Pi.

D. Case Study

To further demonstrate ComHA's capability in handling complex scenarios and validate its practical deployment and application effectiveness, this section presents the implementation and practical outcomes of ComHA in a specific scenario from a Smart Grid Project.

Project Background. The scenario described in this chapter originates from a real-world incident—the tornado disaster that struck Baiyun District, Guangzhou on April 27, 2024. During this event, the smart monitoring system of China Southern Power Grid faced critical challenges: cameras deployed in the disaster area failed to reliably detect transmission facility deformations and foreign object intrusions caused by tornadoes due to algorithmic limitations, resulting in delayed fault localization. To enable rapid damage repair and enhance grid resilience, emergency updates of object detection algorithms were required rapidly in the affected area, enabling adaptation to multi-scale, low-visibility detection demands under extreme weather conditions. However, the existing system presented two key technical bottlenecks:

1) **Model Heterogeneity**: The system integrated 28 neural network models (including FSRCNN, YOLOv5, MobileNet, DeepLabV3, etc.), each optimized for specific hardware architectures (e.g., Huawei Atlas-series NPUs and Nvidia Jetson GPUs);
2) **Inefficient Deployment**: Traditional full-model updates necessitated cross-architecture model conversion (e.g., TensorRT engine deployment) and quantization calibration.

Furthermore, mainstream incremental learning methods (e.g., iCaRL [40], EEIL [41]) required 200+ training epochs to mitigate catastrophic forgetting, rendering them impractical for real-time updates across distributed edge devices. The ComHA

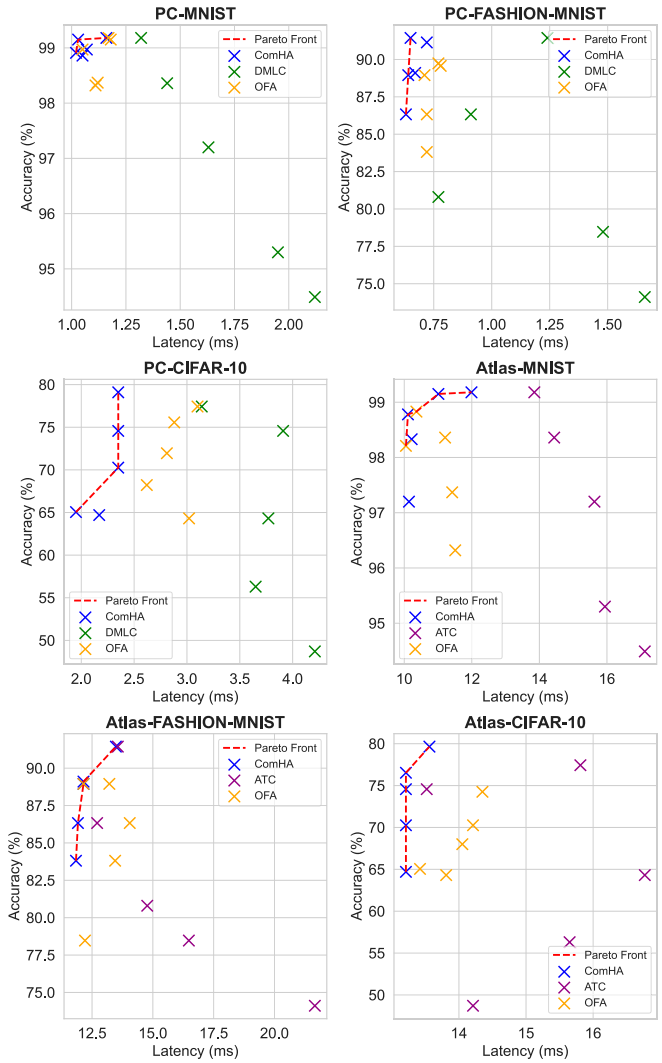

Fig. 6. Scatter plots comparing the models built by ComHA and baseline methods across various datasets (MNIST, FASHION-MNIST, CIFAR-10) and devices (PC and Atlas). The Pareto front for performance-cost trade-off is highlighted.

framework addresses this limitation through a three-phase cloud-edge coordination mechanism: (1) aggregating global historical tornado data to construct a disaster feature knowledge base, (2) employing genetic algorithms in the cloud to dynamically optimize the old-to-new sample ratio during incremental learning processes—a critical factor balancing historical knowledge retention and new scenario adaptation, and (3) packaging the optimized ratio configuration into lightweight "Seed" files that are deployed to edge devices. This approach enables edge-end devices to conduct efficient incremental training with minimal epochs by leveraging cloud-optimized sample distributions, effectively decoupling computationally intensive optimization from resource-constrained edge execution.

In heterogeneous computing ecosystems spanning diverse hardware architectures (e.g., GPU/NPU/CPU hybrids), ComHA enables cross-device model updates within 20 epochs—achieving a 90% training time reduction versus conventional incremental learning approaches (typically requiring 200+ epochs) while preserving competitive accuracy (90%) across all platforms through cloud-optimized sample distribution.

Dataset and Model. Subject to data confidentiality, we replaced the real production data with the public CIFAR-100 as the mock data. This standardized dataset comprises 100 object categories with 600 images per class, providing a balanced evaluation platform for multi-class incremental learning scenarios. The ResNet-18 architecture is employed as the backbone network, trained with a batch size of 64 using the Adam optimizer (initial learning rate: 0.001).

Scenario Setup. The scenario consists of 20 incremental learning sessions, with each session introducing five new classes while retaining 2000 exemplar images from previous classes. To evaluate efficiency, Epochs (ep) is defined, where "n" denotes the number of epochs used for training in each session. The Average Accuracy, as described in [42], is used as a key metric to assess the model's effectiveness in mitigating the catastrophic forgetting (CF) problem.

To enable CED coordination, the system adopts Kubernetes-orchestrated containerized model repositories for version-controlled deployment, coupled with lightweight SDK-encapsulated runtime libraries that facilitate over-the-air transmission of optimized hyper-parameters.

Additional hyperparameter. The old-to-new sample ratio greatly influences the convergence speed of the model by balancing the retention of prior knowledge and the adaptation to new classes as observed in Fig. 8.

To provide a clearer understanding of the concept of the ratio, we offer the following explanation. In incremental learning, each time a new class is introduced, the training sample ratio among all classes (e.g., $A:B:C:D$ for four classes) must be re-defined. This ratio plays a crucial role in balancing the retention of prior knowledge and the adaptation to new classes. Over the course of learning, this ratio evolves into a lower triangular matrix where each row represents the sampling ratios for all classes up to the current incremental step. For instance, the following 4×4 matrix illustrates the ratio representation for incremental learning with four classes:

$$\begin{aligned}
&\text{Initial (1 class):} \begin{bmatrix} a & 0 & 0 & 0 \end{bmatrix} \\
&\text{After adding a second class (2 classes):} \begin{bmatrix} b & f & 0 & 0 \end{bmatrix} \\
&\text{After adding a third class (3 classes):} \begin{bmatrix} c & g & x & 0 \end{bmatrix} \\
&\text{After incremental learning with four classes:} \begin{bmatrix} d & h & y & z \end{bmatrix}
\end{aligned}$$

In our case study, we additionally employ a genetic algorithm in the cloud-level AutoML to optimize this ratio for each learning context. The optimal ratio is then wrapped in the seed and sent to the edge devices for efficient incremental learning. Performance is evaluated against five baseline methods in terms of average model accuracy and training efficiency.

Baseline methods. We compare ComHA with the following widely used and highly effective incremental learning methods in CED scenarios:

1) **iCaRL** [40]: Proposes a class-incremental learning strategy that maintains a small exemplar set of old classes and combines distillation loss with nearest-mean-of-exemplars classification to prevent forgetting.

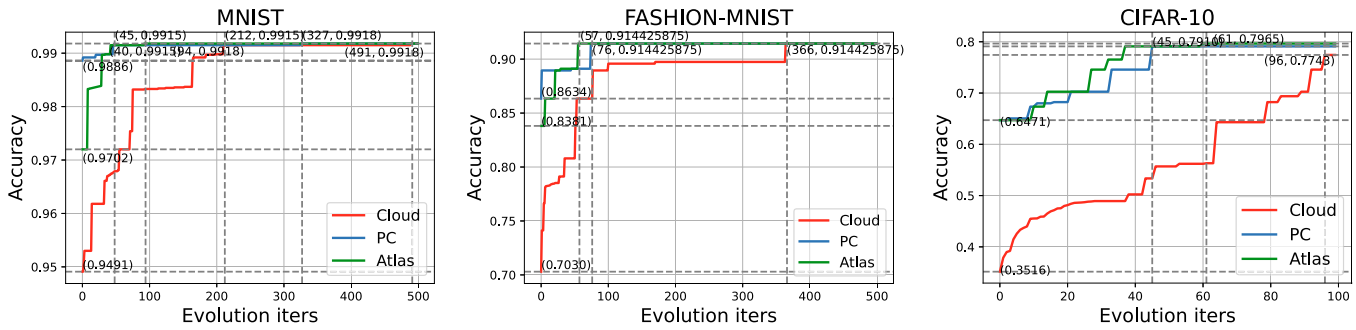

Fig. 7. The evolution of models produced by ComHA in the cloud (stage I) and at the edge (PC and Atlas).

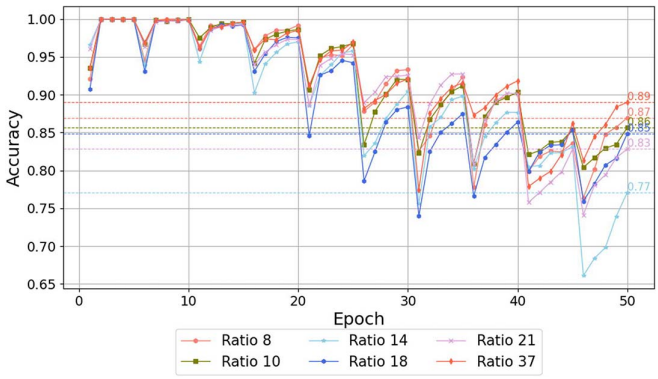

Fig. 8. Case study with incremental learning: the figure clearly demonstrates that different ratios significantly impact the training process and model performance.

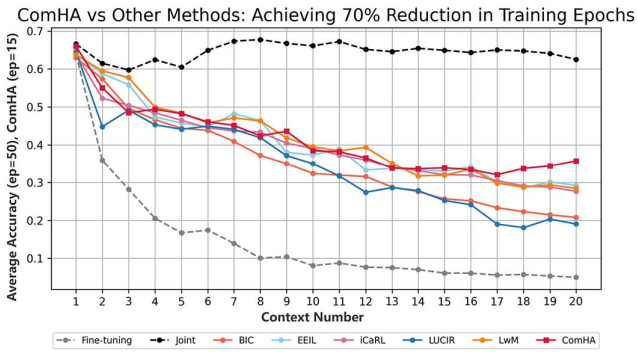

Fig. 9. Performance comparison of ComHA with baseline methods on CIFAR-100. ComHA achieves similar or higher accuracy with only 15 training epochs per context, compared to 50 epochs required by other methods, demonstrating a 70% reduction in training time while maintaining the best average accuracy.

2) **BiC** [43]: Addresses data imbalance between old and new classes by adding a bias correction layer to adjust classifier outputs, mitigating bias towards new classes without storing old class data.
3) **EEIL** [41]: Introduces an end-to-end incremental learning approach that jointly learns data representation and classifier, using distillation loss to retain old class knowledge and cross-entropy loss for new classes, while keeping a small exemplar set.
4) **LwM** [44]: Introduces Attention Distillation Loss to preserve old class information without storing their data, penalizing changes in the classifier's attention maps to prevent catastrophic forgetting.
5) **LUCI** [45]: Tackles class imbalance by incorporating cosine normalization, less-forget constraint, and inter-class separation, rebalancing the training process to improve overall performance.

In Fig. 9, we compare the performance of ComHA with several widely used and highly effective incremental learning methods in CED scenarios. We also include Joint-Training and Fine-Tuning methods for reference. Joint-Training serves as an upper-bound benchmark, training on all available data simultaneously. Fine-Tuning acts as a lower-bound benchmark, updating the model using only new data without any anti-catastrophic forgetting strategies.

In the comparison, ComHA's only takes 15 epochs of training for each context, whereas the baseline methods need 50 epochs to reach a competitive accuracy. This means that ComHA achieves a 70% reduction in training epochs while attaining the highest average accuracy among all methods. This advantage demonstrates ComHA's theoretical speedup, even without considering the heterogeneous performance differences of edge devices in CED environments.

VI. Observations and Analysis

A. Analysis of Model Measurement Indicators

Our first empirical finding is that commonly used model performance indicators often fail to reflect the actual performance of neural network models on heterogeneous devices.

As shown in Fig. 4, the actual inference latency of the model has no clear correlation with FLOPs and Params. For instance, many model instances with nearly identical Params and FLOPs exhibit significant differences in inference latency. Similarly, there is no consistent relationship between the number of layers and the actual performance of the model, although the number of layers does influence the range of latency. These findings highlight the limitations of cloud-based model evaluation and its potential to mislead the AutoML process.

B. Analysis of Model Adaptation

AI model adaptation has a critical impact on performance across heterogeneous devices.

From Fig. 5 and Table I, it is evident that without adaptation, model inference latency on CPUs can be up to four times slower compared to GPUs and NPUs, which renders such latency unacceptable for practical applications. For example, Raspberry Pi devices show particularly slow inference times when adaptation is unavailable. Additionally, adaptation significantly reduces energy consumption during on-device inference. As shown in Table I, inference on PCs after adaptation consumes 2-4W less power compared to non-adapted models. This advantage is especially critical for battery-powered edge devices.

Furthermore, the inference latency of similar device types is correlated. For instance, Fig. 5 shows a latency pattern for Nvidia GPUs that aligns across different GPU models, such as the GeForce GTX 1060 and Tesla T4. However, the heterogeneity of devices, particularly in their architectures, results in strong discrepancies in performance, underscoring the necessity of model localization within a collaborative framework like ComHA.

C. Generalization and Efficiency of ComHA

As shown in Fig. 9, ComHA demonstrates great generalization capabilities in CED scenarios, even under complex tasks and models. Across heterogeneous cloud-edge-device environments, it achieves competitive results with significantly fewer training epochs. ComHA reduces the number of training epochs by 70%, yet maintains superior average accuracy and training efficiency compared to widely adopted methods in both industry and academia. These findings demonstrate ComHA's ability to handle complex scenarios for model training and adaptation, including applications such as power grid systems where edge devices must quickly update models to reflect changes like updated maps.

D. Overall Performance Comparison

ComHA shows clear advantages in adapting models to heterogeneous devices, outperforming mainstream model conversion and optimization methods in accuracy, inference latency, and efficiency.

First, ComHA achieves superior or competitive accuracy while maintaining low latency, as shown in Fig. 6. On datasets such as Atlas-MNIST and Atlas-CIFAR-10, ComHA consistently produces Pareto-optimal models. Even in PC environments, such as PC-MNIST and PC-FASHION-MNIST, ComHA closely approaches the Pareto front, indicating its ability to balance accuracy and latency efficiently. Fig. 6 and Table IV demonstrate that ComHA effectively balances accuracy and latency while exhibiting robust adaptability across various devices and datasets. Although OFA achieves slightly faster inference in some cases, such as Atlas-MNIST, it requires additional adaptation for heterogeneous devices, increasing time and tool dependencies. In contrast, ComHA eliminates such dependencies through cloud-edge-device collaboration, providing a more integrated and efficient solution.

It is worth noting that the memory footprints reported in Table IV show large discrepancies across devices even for identical model topologies generated by ComHA and converted by ATC. This is because GPU-equipped platforms load the full CUDA runtime and its libraries when initializing an AI model, which substantially inflates memory usage. In contrast, energy-constrained devices such as the Atlas200DK and Raspberry Pi preload only a minimal set of dependencies, resulting in much smaller footprints. Moreover, as discussed in Section III-B, conversion tools like ATC often introduce numerical precision loss (e.g., automatic quantization from floating-point to INT8), and unsupported high-performance operators are replaced by device-specific fallbacks. Consequently, beyond memory utilization, significant accuracy degradation and increased inference latency are also inevitable in more complex conversion scenarios.

Second, Fig. 7 illustrates that ComHA can collaboratively train models on edge/end devices to achieve the same performance as cloud-based training, and in some cases even higher accuracy. For example, on the CIFAR-10 dataset, ComHA outperforms the initial cloud-trained model through collaborative training. This demonstrates the framework's effectiveness in refining models locally.

Third, ComHA achieves faster convergence during training, halving the required iterations compared to cloud-only training. The hierarchical design of ComHA——optimizing the search space at the cloud level before refining at the edge——significantly improves both training speed and model quality, making it particularly advantageous for edge computing scenarios.

Fourth, the ComHA framework is also effective in addressing incremental learning scenarios (Figs. 8 and 9). This makes it suitable for handling dynamic and evolving environments such as industrial automation, smart transportation, and power grid systems, where edge devices must quickly adapt models to accommodate rapidly changing conditions including updated maps, task requirements, and resource constraints. This adaptability highlights ComHA's versatility in diverse real-world applications.

Fifth, ComHA can be seamlessly extended to the transfer and local deployment of large-scale models such as modern Large Language Models (LLMs), which are themselves based on the Transformer architecture. Transformer performance is governed by a handful of critical hyperparameters—number of layers, embedding dimension, and number of attention heads—all of which can be automatically optimized via AutoML. For example, the DeepSeek R1 system leverages reinforcement-learning–based AutoML to fine-tune these parameters for personalized model delivery. By contrast, conventional Transformer conversion and migration techniques often degrade performance on heterogeneous hardware—particularly NPUs—due to missing operator support or quantization-induced precision loss. ComHA's hierarchical, precision-preserving pipeline maintains model accuracy and efficiency across the CED spectrum, making it a highly promising approach for future large-model adaptation and deployment in diverse computing environments.

VII. Conclusion

The heterogeneity of devices at the network edge calls for tailored model building that needs to be more flexible than the

traditional model adaptation. In this paper, a cloud-edge-device collaborative model building framework (ComHA) is proposed. Using a hierarchical AutoML design, ComHA leverages the cloud to reduce the search space with diverse algorithms and allows the target device to perform a second stage of AutoML within a small subset of solution space. As a result, edge or end devices are able to find and build the models that optimally match their local environment and local data without explicit model adaptation. Comprehensive experiments show that, compared to existing model building methods, ComHA produces models that excel in accuracy and inference latency. We also demonstrate through empirical study that existing cloud-based evaluation cannot well reflect the actual performance of models at the edge, which further explains the motivation of our work and its advantage in the experiment.

Our future research will be based on ComHA with focus on joint optimization of model building and task scheduling in heterogeneous systems.

REFERENCES

[1] Y. Chen, B. Liu, W. Lin, and H. Cheng, "Survey of cloud-edge collaboration," *Comput. Sci.*, vol. 48, no. 3, p. 259, 2021.

[2] M. Jiang, T. Wu, Z. Wang, Y. Gong, L. Zhang, and R. P. Liu, "A multi-intersection vehicular cooperative control based on end-edge-cloud computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 3, pp. 2459–2471, Mar. 2022.

[3] Z. Yang, B. Liang, and W. Ji, "An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16779–16786, Dec. 2021.

[4] M. Zhu and J. Li, "Blockchain based on reliable task offloading strategy for edge computing in smart home," in *Proc. 8th Int. Conf. Intell. Comput. Signal Process. (ICSP),* 2023, pp. 1364–1368.

[5] Z. Zhou, S. Yu, W. Chen, and X. Chen, "CE-IoT: Cost-effective cloud-edge resource provisioning for heterogeneous IoT applications," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8600–8614, Sep. 2020.

[6] M. García-Domínguez, C. Domínguez, J. Heras, E. Mata, and V. Pascual, "UFOD: An AutoML framework for the construction, comparison, and combination of object detection models," *Pattern Recognit. Lett.*, vol. 145, no. C, pp. 135–140, May 2021.

[7] D. Souza, L. Cabrita, C. Galinha, T. Rato, and M. Reis, "A Spectral AutoML approach for industrial soft sensor development: Validation in an oil refinery plant," *Comput. Chem. Eng.*, vol. 150, Apr. 2021, Art. no. 107324.

[8] X. Shi, Y. D. Wong, C. Chai, and M. Z.-F. Li, "An automated machine learning (AutoML) method of risk prediction for decision-making of autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 11, pp. 7145–7154, Nov. 2021.

[9] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," 2020, *arXiv:2008.06402*.

[10] B. Zhang, T. Xiang, H. Zhang, T. Li, S. Zhu, and J. Gu, "Dynamic DNN decomposition for lossless synergistic inference," 2021, *arXiv:2101.05952*.

[11] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent IoT applications: A cloud-edge based framework," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 35–44, 2020.

[12] S. Zhang, J. Wang, J. Tong, J. Zhang, and M. Zhang, "Cloud-edge fusion based abnormal object detection of power transmission lines using incremental learning," *IEEE Access*, vol. 8, pp. 218694–218701, 2020.

[13] S. Deng, Z. Chen, F. Kuang, C. Yang, and W. Gui, "Optimal control of chilled water system with ensemble learning and cloud edge terminal implementation," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7839–7848, Nov. 2021.

[14] F. Zhang, J. Qin, D. Ran, L. Cao, and P. Guo, "A satellite ground system with an 'edge-cloud-terminal' structure," in *Proc. 6th Int. Conf. Frontiers Educ. Technol. (ICFET)*, 2020, New York, NY, USA: ACM, pp. 217–221.

[15] X. Zhang, Z. Zeng, P. Wang, J. Song, and Z. Kong, "A hybrid edge-cloud computing method for short-term electric load forecasting based on smart metering terminal," in *Proc. IEEE 4th Conf. Energy Internet Energy Syst. Integr. (EI2)*, 2020, pp. 3101–3105.

[16] Z. Ma, H. Li, W. Fang, Q. Liu, B. Zhou, and Z. Bu, "A cloud-edge-terminal collaborative system for temperature measurement in COVID-19 prevention," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6.

[17] Y. Qiao, Q. Sun, H. Cao, J. Wang, and T. Hao, "Privacy-preserving dish-recommendation for food nutrition through edging computing," *Trans. Emerg. Telecommun. Technol.*, vol. 33, p. 6, Jan. 2020.

[18] X. Yang, S. Wang, J. Dong, J. Dong, M. Wang, and T.-S. Chua, "Video moment retrieval with cross-modal neural architecture search," *IEEE Trans. Image Process.*, vol. 31, pp. 1204–1216, 2022.

[19] C.-W. Tsai and Z.-Y. Fang, "An effective hyperparameter optimization algorithm for DNN to predict passengers at a metro station," *ACM Trans. Internet Technol.*, vol. 21, no. 2, pp. 1–24, Mar. 2021.

[20] Z. Czako, G. Sebestyen, and A. Hangan, "AutomaticAI – A hybrid approach for automatic artificial intelligence algorithm selection and hyperparameter tuning," *Expert Syst. Appl.*, vol. 182, no. C, pp. 1–10, Nov. 2021.

[21] A. M. Alanezi, K. Hallinan, and K. Huang, "Automated residential energy audits using a smart wifi thermostat-enabled data mining approach," *Energies*, vol. 14, no. 2500, p. 4, 2021.

[22] J. Parker-Holder et al., "Automated reinforcement learning (AutoRL): A survey and open problems," 2022, *arXiv:2201.03916*.

[23] R. Upadhyay, R. Phlypo, R. Saini, and M. Liwicki, "Sharing to learn and learning to share - Fitting together meta-learning, multi-task learning, and transfer learning : A meta review," 2021, *arXiv:2111.12146*.

[24] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, 2019.

[25] X. Liu, Y. Zhao, M. Lu, Z. Chen, and S. Huang, "Multi-objective optimization for a dual-flux-modulator coaxial magnetic gear with double-layer permanent magnet inner rotor," *IEEE Trans. Magn.*, vol. 57, no. 7, pp. 1–5, Jul. 2021.

[26] D. Karl et al., "AutoAIViz: Opening the blackbox of automated artificial intelligence with conditional parallel coordinates," 2019, *arXiv:1912.06723*.

[27] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," 2018. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.pdf

[28] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018. [Online]. Available: https://arxiv.org/pdf/1812.00332

[29] B. Lu, J. Yang, W. Jiang, Y. Shi, and S. Ren, "One proxy device is enough for hardware-aware neural architecture search," in *Proc. ACM Meas. Anal. Comput. Syst.*, Dec. 2021, vol. 5, pp. 1–34.

[30] C.-J. Wu et al., "Machine learning at Facebook: Understanding inference at the edge," in *IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 331–344.

[31] W. Shen, W. Lin, Y. Wu, F. Shi, W. Wu, and K. Li, "Evolving deep multiple kernel learning networks through genetic algorithms," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1569–1580, Feb. 2023.

[32] H. Zeng, W. Shen, H. Wu, M. Dong, W. Lin, and C. L. P. Chen, "CAGO-ECIL: Cloud-assisted genetic optimization for edge-class incremental learning with training acceleration," *Future Gener. Comput. Syst.*, vol. 174, Jan. 2026, Art. no. 108021.

[33] M. Lindauer and F. Hutter, "Best practices for scientific research on neural architecture search," 2019, *arXiv:1909.02453*.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Red Hook, NY, USA, Curran Associates, 2012, vol. 1, pp. 1097–1105.

[35] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive D2D collaboration for energy-efficient mobile edge computing," 2017, *arXiv:1703.10340*.

[36] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," 2018, *arXiv:1807.11626*.

[37] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," 2016, *arXiv:1611.06440*.

[38] ysh329. "Deep learning model convertors." GitHub. Accessed Sep. 10, 2024. [Online]. Available: https://github.com/ysh329/deep-learning-model-convertor

[39] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://dblp.org/rec/conf/iclr/CaiGWZH20.html
[40] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA: IEEE Press, Jul. 2017, pp. 5533–5542.
[41] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," Sep. 2018, *arXiv:1807.09536 [cs]*.
[42] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. Van De Weijer, "Class-incremental learning: Survey and performance evaluation on image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 5513–5533, May 2023.
[43] Y. Wu et al., "Large scale incremental learning," May 2019, *arXiv:1905.13260 [cs]*.
[44] P. Dhar, R. V. Singh, K.-C. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," Apr. 2019, *arXiv:1811.08051 [cs]*.
[45] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, "Learning a unified classifier incrementally via rebalancing," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit. (CVPR)*, Long Beach, CA, USA: IEEE Press, Jun. 2019, pp. 831–839.

Weiwei Lin (Senior Member, IEEE) received the B.S. and M.S. degrees from Nanchang University, in 2001 and 2004, respectively, and the Ph.D. degree in computer application from the South China University of Technology, in 2007. He has been serving as Visiting Scholar with Clemson University from 2016 to 2017. He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology. His research interests include distributed systems, cloud computing, big data computing, and AI application technologies. He has published more than 150 papers in refereed journals and conference proceedings.

Wangbo Shen received the B.S. degree from Changsha University, Changsha, China, in 2012 and 2016, and the M.S. degree from the Central South University, Changsha, China in 2016 and 2019 respectively. He is currently working toward the Ph.D. degree with the Department of Computer Application, South China University of Technology, Guangzhou, China supervised by Dr. Weiwei Lin. His research interests include Kernel learning, AutoML, and edge computing.

Wentai Wu (Member, IEEE) received the bachelor's and master's degrees from the South China University of Technology, in 2015 and 2018, respectively. Since 2024, he is an Associate Professor with the Department of Computer Science, College of Information Science and Technology, Jinan University. His research interests include distributed intelligent systems and sustainable computing. He has published over 30 refereed papers and contributed to two books on relevant subjects.

Keqin Li (Fellow, IEEE) received the B.S. degree in computer science from Tsinghua University, in 1985, and the Ph.D. degree in computer science from the University of Houston, in 1990. He is a SUNY Distinguished Professor with the State University of New York and a National Distinguished Professor with Hunan University (China). He has authored or co-authored more than 1200 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top few most influential scientists in parallel and distributed computing, regarding single-year impact (ranked 2nd) and career-long impact (ranked 3rd) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars (2023–2025) and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in ScholarGPS Highly Ranked Scholars (2022–2024) and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He is a member of the SUNY Distinguished Academy. He is an AAAS Fellow, AAIA Fellow, ACIS Fellow, and AIIA Fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe).