End-Edge-Cloud Collaborative Computing for Deep Learning: A Comprehensive Survey

Yingchao Wang, Chen Yang, *Member, IEEE,* Shulin Lan, *Member, IEEE,* Liehuang Zhu, *Senior Member, IEEE,* Yan Zhang, *Fellow, IEEE*

Abstract—**The booming development of deep learning applications and services heavily relies on large deep learning models and massive data in the cloud. However, cloud-based deep learning encounters challenges in meeting the application requirements of responsiveness, adaptability, and reliability. Edge-based and end-based deep learning enables rapid, near real-time analysis and response, but edge nodes and end devices usually have limited resources to support large models. This necessitates the integration of end, edge, and cloud computing technologies to combine their different advantages. Despite the existence of numerous studies on edge-cloud collaboration, a comprehensive survey for end-edge-cloud computing-enabled deep learning is needed to review the current status and point out future directions. Therefore, this paper: 1) analyzes the collaborative elements within the end-edge-cloud computing system for deep learning, and proposes collaborative training, inference, and updating methods and mechanisms for deep learning models under the end-edge-cloud collaboration framework. 2) provides a systematic investigation of the key enabling technologies for end-edge-cloud collaborative deep learning, including model compression, model partition, and knowledge transfer. 3) highlights six open issues to stimulate continuous research efforts in the field of end-edge-cloud deep learning.**

Index Terms—**Deep learning, deep neural networks, edge computing, cloud computing, end-edge-cloud collaboration, end-edge-cloud computing**

I. Introduction

DEEP learning (DL) attempts to mimic the human brain using multi-layer neural networks to learn representations of data with multiple levels of abstraction [1]–[3]. In recent years, with the explosive growth in data volume and computational capabilities, DL has achieved groundbreaking progress in image, video, speech, and audio processing [4] and has played a pivotal role in propelling advancements in artificial intelligence (AI) applications across diverse domains, including healthcare and smart home [5], autonomous driving [6], smart city [7], and smart manufacturing [8]. However, in practical applications, the effectiveness of a deep learning model (DLM) is often enhanced with the size of the model, which is indicated by the number of network layers and model parameters. For example, the LeNet proposed in 1998 consisted of only 5 layers, while the ResNet introduced in 2015 featured 152 layers. Larger DLMs possess the ability to learn high-level abstract features from complex and high-dimensional data, leading to improved feature extraction and inference performance. However, this advancement comes at the cost of a rising need for computing and storage resources. For example, even the VGG-16 network, with 138 million parameters, requires approximately 1.6 billion floating-point operations per second for processing and consumes around 528 MB of memory. More recently, increasingly complex DLMs, especially the large pre-trained foundation models such as GPT-3 [9], "Yuan 1.0" [10] and PaLM [11], pose significant demands on computing resources as depicted in Table I. For example, "Yuan 1.0" [10], with 245 billion parameters and a 5TB training dataset, was trained for 16 days on a computing power platform of 1028 GPUs.

Such substantial storage and computing resource requirements greatly limit the application of large DLMs on edge nodes or end devices with limited computational capabilities. For example, the Hi3516E V200 SoC image processing chip in the edge device, equipped with an Arm Cortex A7 CPU, only possesses 512MB Onchip and 1GB Offchip memory [12], which cannot efficiently train and execute large DLMs such as VGG-16 (approximately 528MB) and VGG-19 (about 548MB). Consequently, cloud centers with robust computing power, extensive storage capacity, and high data throughput have become the primary platform for large DL applications. However, cloud-based DL encounters certain challenges and issues:

1) Response time of decision-making: The response time of DL applications depends on the computation delay and the communication delay. The computation delay is affected by factors such as the size of DLMs, available computing power, and the degree of task parallelism. Even if the cloud centers have sufficient computing resources and mature parallelization technologies for DLMs, the transmission of data between the users' devices and cloud servers introduces unavoidable communication delays. Limited by current communication technologies, wide area networks inevitably suffer from high latency and unstable performance issues since it was originally designed to increase bandwidth capacity and link efficiency [13]. Moreover, the exponential growth of multi-modal data

This work was supported by the National Key R&D Program of China under Grant 2021YFB1715700; in part by the National Natural Science Foundation of China under Grant 62103046, 72201266, 72192843, and 72192844; in part by the Fundamental Research Funds for the Central Universities under Grant E1E40805X2 and 2023CX01020. (*Corresponding author: Chen Yang, Shulin Lan.*)

Yingchao Wang, Chen Yang, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. (e-mail: yingchaowang@bit.edu.cn; yangchen666@bit.edu.cn; liehuangz@bit.edu.cn)

Shulin Lan is with the School of Economics and Management, University of the Chinese Academy of Sciences, Beijing, China. (e-mail: lanshulin@ucas.ac.cn)

Yan Zhang is with the Department of Informatics, University of Oslo, Norway, and also with Simula Metropolitan Center for Digital Engineering, Norway (e-mail: yanzhang@ieee.org)

TABLE I
RECENT TYPICAL LARGE PRE-TRAINED DEEP LEARNING MODEL

Model	Time	Size (B)	Hardware	Training day
T5	2019	11	1024 TPU v3	N/A
GPT-3	2020	175	10000 GPUs	30
GShard	2020	600	2048 TPU v3	4
Yuan 1.0	2021	245	2128 GPUs	16
GLaM	2021	1200	1024 TPU v4	23.9
Gopher	2021	280	4090 TPU v3	38.3
LaMDA	2022	137	1024 TPU v3	57.7
PaLM	2022	540	6144 TPU v4	50
U-PaLM	2022	540	512 TPU v4	5
GPT-4	2023	N/A	N/A	N/A
PanGu-Σ	2023	1085	512 Ascend 910	100

generated by the proliferation of sensors and Internet of Things (IoT) devices is putting tremendous pressure on the network. This data can amount to hundreds of millions of megabytes per second. For example, Google's self-driving car alone can generate up to 750 megabytes of sensor data every second [14]. Uploading such colossal volumes of data to cloud servers will exert significant pressure on the core network and exacerbate communication delays.

2) Lifelong-learning and self-adaptation: Mobile DL applications, such as unmanned vehicles and drones, need to continuously interact with dynamic environments, along with the ability to self-learn and self-adapt their behavior based on perception and feedback. However, the adoption and updating of cloud-based DLMs requires frequent interactions between mobile devices, such as vehicles, and the cloud center, resulting in increased energy consumption for these mobile devices. Furthermore, unreliable networks may cause transmission interruptions or errors, significantly affecting the lifelong learning, adaptation processes, and overall robust operations of industrial DL applications.

3) Data security and privacy: The traditional cloud computing paradigm involves transmitting raw or pre-processed data to the cloud for storage and analysis. Despite numerous studies on encrypted communication and computing technologies, their widespread application and deployment have been constrained by the high costs associated with encryption and decryption. Moreover, data owners are increasingly prioritizing data privacy and security, leading to their reluctance to allow personal data to leave the local area. Therefore, decentralization and the emphasis on processing at or near the data sources whenever feasible become imperative.

Confronted with these challenges and issues, the migration of DLMs to edge nodes or end devices close to data sources, known as edge AI, emerges as an effective solution. This approach aims to significantly reduce service response time and alleviate the impact of data uploading on the network while enhancing data privacy and improving DLM's adaptability [15], [16]. Numerous studies have explored various methods for deploying DLMs on edge or end devices. One approach involves deploying DLMs on a single edge or end device without compromising performance and accuracy. This includes model compression and acceleration [17], [18], lightweight model design [19], and hardware design and optimization [20]. Another approach leverages the collective resources of multiple edge nodes or end devices to handle DL tasks and overcome the limitations of individual devices, such as federated learning (FL). However, the computing, storage, energy, and communication resources of edge and end devices are often limited. Meeting the diverse and stringent requirements of various applications remains challenging. Moreover, for large-scale models, especially pre-trained foundation models, the powerful computing power provided by cloud centers is still indispensable. As a result, the computing paradigm of DL has evolved toward end-edge-cloud computing.

Compared to relying solely on cloud computing or edge computing, collaborative end-edge-cloud computing facilitates real-time, fast-response, dynamic, scalable, and secure DL services across different time and space scales, effectively catering to the diverse requirements of DL applications [8]. Therefore, it has become a prominent research topic in both academia and industry. While the end-edge-cloud computing approach offers several advantages, certain issues and obstacles need to be addressed to fully harness the potential of end-edge-cloud collaborative DL:

- What are the essential elements of collaboration?
- How can effective collaboration be achieved?
- Which key technologies play a pivotal role?
- What are the challenges and research directions?

To answer the above questions and unlock the full potential of DL in distributed end-edge-cloud computing environments, this survey systematically analyzes key collaborative elements and proposes modes and mechanisms of end-edge-cloud collaborative DL, including co-training, co-inference, and co-updating. Then, it highlights technological advances and challenges, setting the stage for future research in end-edge-cloud computing for DL.

A. Comparison and Our Contribution

1) AI for end-edge-cloud computing: AI, especially DL, presents the potential to provide optimal solutions for key problems and applications within the end-edge-cloud computing framework, such as computing offloading, resource allocation, and security and privacy. These aspects have been extensively reviewed in various surveys [21]–[26]. For example, Hua et al. [21] comprehensively examined existing studies on edge computing performance optimization and different application scenarios of AI. In these surveys, AI or DL plays a pivotal role as an enabling technology in end-edge-cloud computing, effectively functioning as AI for end-edge-cloud computing. In contrast, this paper delves into the comprehensive lifecycle of AI models, particularly DLMs, within the end-edge-cloud computing framework, which is known as end-edge-cloud computing for AI or DL. This lifecycle encompasses model training, inference, and updating that involves dynamic adaptation of DLMs to real-time environmental changes.

2) End-edge-cloud computing for AI: Deploying AI models, especially DLMs, on resource-constrained end devices or edge servers (known as edge-AI), such as smartphones, IoT devices, and embedded systems, is an essential aspect and key technology within the end-edge-cloud computing collaboration

TABLE II
COMPARISON WITH RELATED WORK

Category	Ref.	Time	Key contributions and concerns	Elements and mechanisms of end-edge-cloud DL	Model compression technologies	Model partitioning technologies	Knowledge transfer technologies
An AI or DL model on an end or edge node	[27]	2021	Compression technologies and applications for IoT.	✘	✔	✘	✘
	[31]	2021	Compression technologies, tools, frameworks, and hardware for machine learning on edge.	✘	✔	✘	✘
	[20]	2022	Hardware accelerators and model compression technologies for DL on portable devices.	✘	✔	✘	✘
	[29]	2023	Reformable TinyML solutions including identified deployment schemes and available tools.	✘	✘	✘	✘
	[30]	2023	DLM efficiency from modeling techniques to hardware support.	✘	✔	✘	✘
Distribute an AI or DL model on multiple ends or edge nodes	[33]	2019	Various approaches for DL inference across end-edge-cloud, and methods for training DLMs across multiple edge devices.	Distributed edge training and co-inference	✔	✔	✘
	[34]	2019	Key building blocks of edge machine learning, different neural network architectural splits and theoretical and technical enablers.	Distributed edge training and inference	✘	✔	✘
	[35]	2020	Application scenarios of DL for edge and DL on edge, and the practical implementation methods and enabling technologies for DL on edge.	Edge FL and edge inference	✘	✔	✘
	[36]	2021	Emerging technologies for AI models regarding inference and training on edge	Distributed training and edge co-inference	✔	✔	✘
	[37]	2021	Technical fundamentals, challenges and relevant solutions of decentralized DL.	FL	✘	✘	✘
	[32]	2021	Enabling technologies for edge learning, including model training, inference, security guarantee, privacy protection, and incentive mechanism	Distributed edge training and inference.	✘	✘	✘
	[38]	2022	Architectures, technologies, frameworks and implementations in edge training and inference	Edge FL and co-inference	✔	✔	✘
Distribute an AI or DL model on end-edge-cloud nodes	[41]	2022	Split computing and early exiting strategies for DLMs	✘	✘	✔	✘
	[39]	2023	Collaborative learning mechanism for cloud and edge modeling and advanced edge AI topics including pretraining models, graph neural networks and reinforcement learning	FL and efficiency-primary edge-cloud collaboration	✔	✔	✘
	[40]	2023	Distributed AI empowered by end-edge-cloud computing, including computing paradigms, fundamental and optimization technologies, security and privacy threats and defenses, and applications.	Distributed training and inference	✔	✔	✘
DLMs on end-edge-cloud nodes	This paper	N/A	Elements and mechanisms, key technologies, and challenges and future research of end-edge-cloud collaborative DL	End-edge-cloud co-training, co-inference, and co-updating	✔	✔	✔

framework. On the one hand, several surveys [20], [27]–[31] have focused on efficient DL technologies such as DLM compression and acceleration, lightweight DLM design, and hardware design to optimize DLMs for resource-constrained individual edge or end devices. Providing a summary of recent prominent efficient DL technologies will furnish readers with a succinct grasp of research concepts and trends in this domain.

On the other hand, some studies [32]–[38] have explored distributed AI or DL, particularly distributed training and FL across multiple edge nodes or end devices. For example, Zhang et al. [32] provided an overview of the enabling technologies for edge learning, including model training, inference, security guarantee, privacy protection, and incentive mechanism. While they focused primarily on promoting collaboration within edge computing environments, these surveys occasionally touch upon methods for edge-cloud collaboration, which can provide valuable insights to enhance collaborative DL across the end-edge-cloud spectrum. Some insights in these reviews may be similar to this paper, but they do not systematically consider DL in the end-edge-cloud computing environment.

Regarding AI or DL in the end-edge-cloud computing context, the most relevant surveys are [39]–[41]. Yao et al. [39] conducted a comprehensive survey encompassing both cloud and edge AI. It gave the classification of edge-cloud collaborative learning including privacy-primary collaboration (such as FL) and efficiency-primary collaboration (such as Auto-split [12] and EdgeRec [42]), and discussed potentials and practical experiences of some ongoing advanced edge AI topics including pre-trained models, graph neural networks and reinforcement learning. However, this survey does not elaborate on end-edge-cloud collaborative updating and knowledge sharing, and it refrained from delving deeply into the technical intricacies of end-edge-cloud collaborative learning.

Duan et al. [40] conducted a review of distributed AI empowered by end-edge-cloud computing, encompassing computing paradigms, fundamental and optimization technologies, security and privacy threats and defenses, as well as applications. However, it focused on the end-edge-cloud collaboration for a single AI model, which involves distributed training and inference of an AI model, such as FL and early exiting. Collaboration and knowledge sharing among multiple AI models (e.g. Ding et al. [43] deployed CloudCNN on the cloud server to assist in training the EdgeCNN deployed on the edge server by sharing soft targets and model parameters.) are rarely explored. Meanwhile, it derived a taxonomy for optimization technologies to conduct distributed training and inference, in which model partitioning and compression technologies are categorized into distributed training. While it described the ideas and strategies of the optimizing techniques, there were comparatively fewer discussions of technical details.

Matsubara et al. [41] investigated split computing and early exiting approaches, intending to minimize end-to-end inference latency while closely maintaining model accuracy comparable to that of the original large model. However, these two techniques are specifically designed to facilitate end-edge-cloud collaborative computing for a single model. Techniques such as transfer learning and knowledge distillation need to be added to cover multi-model collaboration and knowledge sharing within the end-edge-cloud framework. Additionally, as the main subject of study [41] was not end-edge-cloud collaboration, it did not investigate the end-edge-cloud collaborative elements and mechanisms, which could offer valuable insights into a broader context of collaborative AI.

While certain studies have reviewed transfer learning [44], [45] and knowledge distillation [46], [47], their exploration did not emphasize the integration of these technologies with end-edge-cloud computing. Furthermore, in our previous studies [8], [48], we have conducted preliminary analyses of end-edge-cloud collaborative elements and mechanisms related to end-edge-cloud co-learning. However, these studies primarily focused on co-inference and did not delve into the discussion of the key technologies involved in this context.

In this study, we present a comprehensive comparison of our paper with recent related studies, as shown in Table II. The analysis reveals several main gaps in the current research landscape, which are summarized as follows. (i) While AI for end-edge-cloud computing and edge-AI have garnered considerable attention, end-edge-cloud collaborative computing for DL, is still in its nascent stages. (ii) Existing studies on end-edge-cloud computing for AI or DL primarily focus on edge-cloud or end-edge-cloud collaboration frameworks for a single AI or DL model, neglecting end-edge-cloud computing for multi-models. This results in an incomplete categorization of end-edge-cloud collaborative DL approaches. (iii) While most studies have primarily outlined the conceptual aspects of AI on end-edge-cloud, there is a pressing need for in-depth exploration and elaboration on these technologies to effectively support end-edge-cloud computing empowered DL applications. (iv) Existing research has not yet fully harnessed the significant potential of knowledge transfer technologies, such as knowledge distillation and transfer learning, and their integration with the end-edge-cloud computing paradigm.

Overall, a systematic survey on end-edge-cloud computing for DL has yet to be conducted. Therefore, this paper aims to address these gaps comprehensively by thoroughly analyzing key elements and mechanisms of end-edge-cloud collaborative DL, highlighting advances in end-edge-cloud deep learning technologies, and identifying challenges and future research directions in this area.

This survey can serve as a valuable resource for researchers, facilitating a quick grasp of recent advances and offering valuable insights into this evolving field of study. The key contributions of this paper are summarized as follows.

- We offer a pioneering analysis of the collaborative learning elements and mechanisms within the realm of end-edge-cloud DL, encompassing the dimensions of data, model (algorithm), and computing power.
- We derive a comprehensive taxonomy for state-of-the-art collaborative learning within the lifecycle of DL. This taxonomy encompasses end-edge-cloud co-training, co-inference, and co-updating, aiming to provide a thorough and detailed description and discussion of existing solutions, with a special focus on knowledge transfer for multiple DLMs on the end-edge-cloud.
- We present and discuss key technologies that enable end-edge-cloud collaborative learning. These include model compression, model partitioning, knowledge distillation, and transfer learning, accompanied by insightful lessons learned.
- We identify and highlight six research challenges and open issues that need to be addressed to enhance the performance of end-edge-cloud collaborative DL. These findings pave the way for future research in the domain of end-edge-cloud computing and its mature applications.

B. The Organization of the Survey

The organization of this paper is illustrated in Figure 1. Section II presents an overview of end-edge-cloud collaborative DL from the dimensions of data, model (algorithm), and computing power, which answers the question of “What are the essential elements of collaboration?”. Based on these identified collaborative elements, Section III derives a taxonomy for end-edge-cloud collaborative DL and elaborates on the mechanisms of each collaborative paradigm, which answers the question of “How can effective collaboration

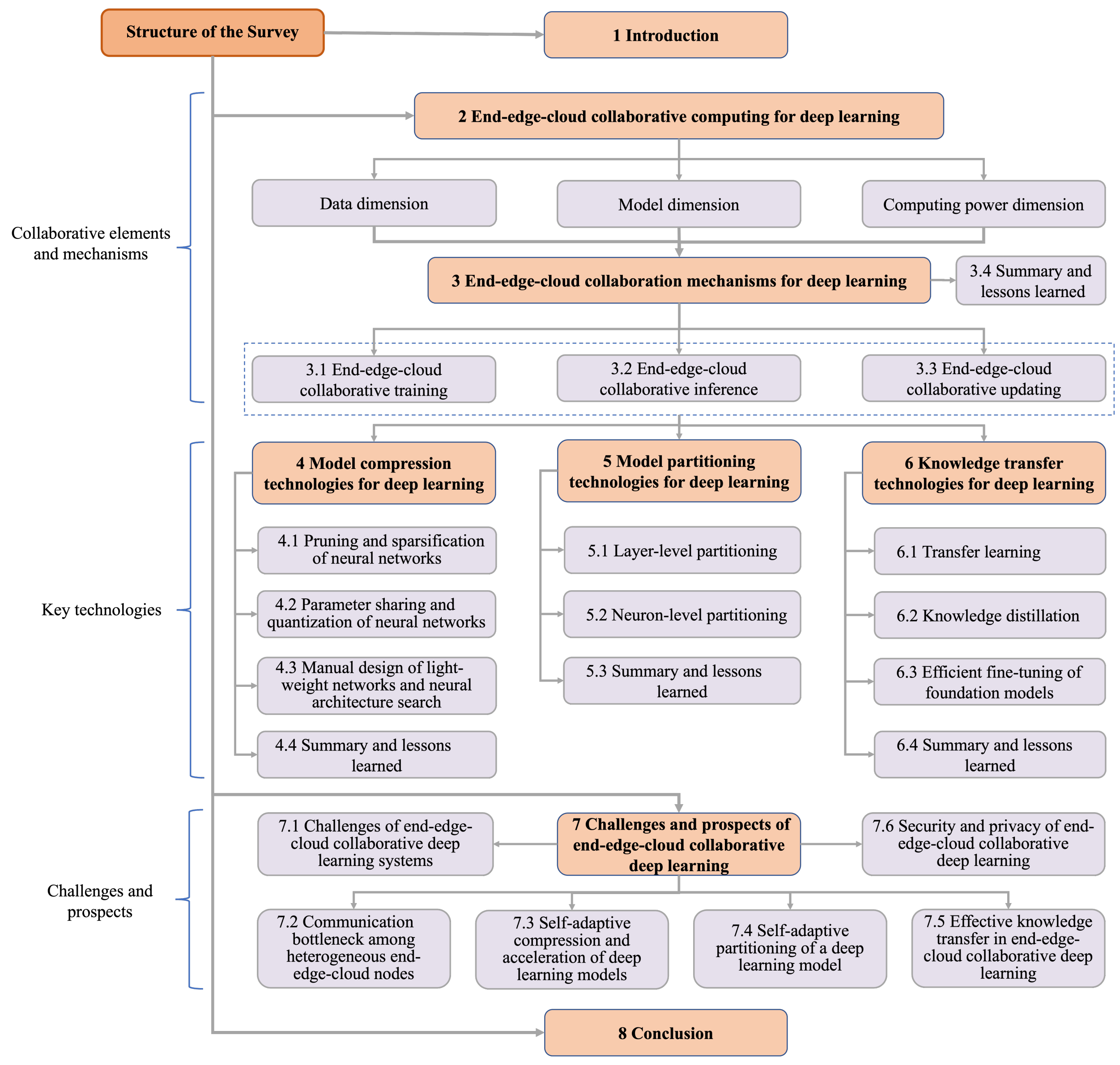
Structure of the Survey
1 Introduction
Collaborative elements and mechanisms
2 End-edge-cloud collaborative computing for deep learning
Data dimension
Model dimension
Computing power dimension
3 End-edge-cloud collaboration mechanisms for deep learning
3.4 Summary and lessons learned
3.1 End-edge-cloud collaborative training
3.2 End-edge-cloud collaborative inference
3.3 End-edge-cloud collaborative updating
Key technologies
4 Model compression technologies for deep learning
4.1 Pruning and sparsification of neural networks
4.2 Parameter sharing and quantization of neural networks
4.3 Manual design of light-weight networks and neural architecture search
4.4 Summary and lessons learned
5 Model partitioning technologies for deep learning
5.1 Layer-level partitioning
5.2 Neuron-level partitioning
5.3 Summary and lessons learned
6 Knowledge transfer technologies for deep learning
6.1 Transfer learning
6.2 Knowledge distillation
6.3 Efficient fine-tuning of foundation models
6.4 Summary and lessons learned
Challenges and prospects
7 Challenges and prospects of end-edge-cloud collaborative deep learning
7.1 Challenges of end-edge-cloud collaborative deep learning systems
7.6 Security and privacy of end-edge-cloud collaborative deep learning
7.2 Communication bottleneck among heterogeneous end-edge-cloud nodes
7.3 Self-adaptive compression and acceleration of deep learning models
7.4 Self-adaptive partitioning of a deep learning model
7.5 Effective knowledge transfer in end-edge-cloud collaborative deep learning
8 Conclusion

Fig. 1. Overall organization of the paper.

be achieved?". The taxonomy includes end-edge-cloud co-training, co-inference, and co-updating. For the above collaborative paradigms, Sections IV-VI summarize and discuss the corresponding key technologies, which answers the question of "Which key technologies play a pivotal role?". It's important to note that the end-edge-cloud collaboration mechanisms discussed in Section III have the potential to leverage any of the technologies detailed in Sections IV-VI. This synergy arises from the compatibility and complementary nature of these technologies. For example, the compressed model can be trained based on transfer learning in a distributed manner, while this distribution can be achieved through model partitioning technologies. Section VII highlights notable challenges and potential research directions which answers the question of "What are the challenges and research directions". Section VIII concludes this paper. The common abbreviations are presented in Table III.

II. End-Edge-Cloud Collaborative Computing for Deep Learning

The fundamental pillars underpinning DL applications encompass data, model (algorithm), and computing power. Therefore, the enabling elements for end-edge-cloud collaborative DL can be analyzed from these dimensions, as shown in Figure 2.

A. Data Dimension

Within the context of knowledge management, the intricate process of perceiving and comprehending objective phenomena unfolds systematically across three interconnected stages: Data, Information, and Knowledge.

1) Data-based end-edge-cloud collaboration: In the realm of DL applications, "Data" refers to raw and unprocessed observations and measurements that are collected from various sources such as IoT sensors or the Internet. "Data", by itself,

TABLE III
LIST OF COMMON ABBREVIATIONS

Abbreviation	Description
DL	Deep Learning
AI	Artificial Intelligence
DLM	Deep Learning Model
IoT	Internet of Things
FL	Federated Learning
FedAvg	Federated Average
Non-IID	Non-Independently and Identically Distributed
FCL	Federated Continual Learning
CPN	Computing Power Network
NAS	Neural Architecture Search
DAG	Directed Acyclic Graph
6G	Sixth-Generation mobile communication technology
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GNN	Graph Neural Network
DNN	Deep Neural Network
DLaaS	Deep Learning-as-a-Service

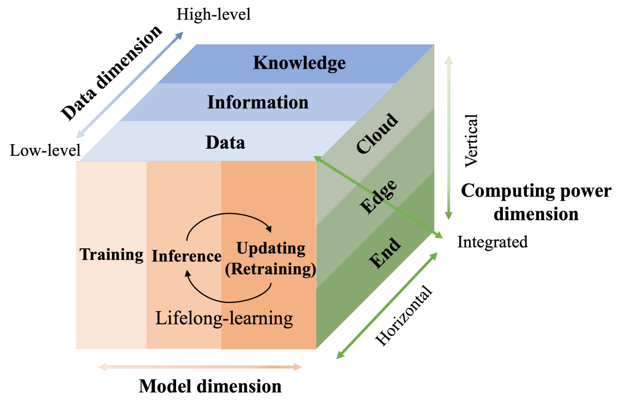

Fig. 2. Analysis model for end-edge-cloud collaborative deep learning.

lacks context and meaning, making it challenging to answer a question or support decision-making.

Data-based collaboration entails the direct uploading of collected raw data from a local device or server to other end devices, edge nodes, or cloud servers for further processing such as training or inference. This approach facilitates efficient sharing and utilization of raw data across various devices and servers within a network. It is notably prevalent in cloud-based DL applications.

2) Information-based end-edge-cloud collaboration: "Information" represents computationally processed and well-organized data imbued with logical relationships and contextual relevance. It empowers us to draw inferences and discern patterns. According to Shannon, information serves as a strictly quantitative metric for measuring communication exchanges, crucial for reducing uncertainty. In DL, information materializes as intermediate data during both training and inference processes, such as gradients during the training phase. Additionally, information can be acquired through data preprocessing and analysis, encompassing tasks such as data cleaning, transformation, combination, mining [49], encryption, and compression.

Information-based collaboration entails the sharing of such processed data across the spectrum of end-edge-cloud nodes. This manifests in various forms, including the transfer of compressed or encrypted data from end devices to edge or cloud servers for further processing, the exchange of intermediate data generated by end-DLMs with edge or cloud-DLMs for subsequent inference, and the collaborative sharing of model gradients to facilitate end-edge-cloud distributed training.

3) Knowledge-based end-edge-cloud collaboration: "Knowledge" encompasses facts, principles, and interpretations of system operations acquired through information processing or learning. It signifies the fusion of information with existing knowledge and experiences, resulting in profound insights, well-informed decision-making, and the competence to effectively apply this knowledge in diverse scenarios. In DL, knowledge can be categorized into various types, including instance knowledge, data representation knowledge, data relational knowledge, model relational knowledge, feature knowledge, and model structured knowledge. These forms of knowledge constitute our comprehension of hidden information, patterns, correlations, and trends within the data, which will be discussed in Section III-C.

Knowledge-based collaboration signifies the sharing of the knowledge as mentioned above across the end-edge-cloud nodes. To illustrate, consider combining with knowledge distillation technology, where smaller student models can be deployed on end or edge nodes while larger teacher models residing in the cloud can oversee the training of these student models by leveraging the teacher DLMs' output feature knowledge, intermediate feature knowledge, relationship knowledge, and structural knowledge. This facilitates end-edge-cloud collaborative updating, as elaborated in Section III-C.

The exchange of information and knowledge serves to reduce data transmission overhead among end-edge-cloud nodes and, to a certain extent, safeguards data privacy [8], [48]. For example, intermediate data, such as feature maps of DLMs can be considerably smaller in size than the original data, thereby mitigating the risk of private data leakage.

B. Model Dimension

Given the dynamic nature of the DL application environment, characterized by the continual arrival of new data and tasks, the concept of the DL lifecycle evolves into one of lifelong or continuous learning, typically encompassing three main phases: training, inference, and updating, as depicted in Figure 2.

Training, including pre-training in large foundation models, refers to the process of adjusting the parameters of a DLM using a dataset so that the model can learn complex representations and functions from the data and make inferences for a specific task [1]. The existing training methods are mostly based on given datasets [4], [50], [51]. In these approaches, the tasks (e.g., membership classes in a classification task) and datasets are predefined and remain unchanged throughout both the training and inference processes. However, DL systems operating in the real world are exposed to continuous streams of data and are required to continuously learn and remember multiple tasks from dynamic data distributions [52], [53]. While training a model on a new large dataset that consists of new and old datasets from scratch is a pragmatic approach, it can be resource-intensive and time-consuming, particularly when dealing with extensive and diverse datasets. Further training the deployed DL model with only the new data is an alternative promising solution, but it faces the significant challenge of catastrophic forgetting, where previously acquired knowledge fades when assimilating new and unfamiliar data observations [54]. Recent techniques such as incremental learning [55] and lifelong learning [50], [52], [56] have focused their efforts on solving the catastrophic forgetting problem and enabling DL to continuously learn and update from incrementally available heterogeneous data.

Given that the traditional training concept doesn't involve catastrophic forgetting and continuous learning, and to highlight that DL is an evolutionary, constantly learning, and adapting process, we employ the broader term "updating" to encompass continuous learning such as incremental learning and lifelong learning, setting it apart from the conventional notions of "training". Naturally, the "updating" stage occurs after the DL model is trained in the lab and deployed in the production environment. In summary, following the initial training and inference, DL undergoes a continuous updating phase, encompassing the full life cycle of DL.

In the traditional cloud-based DL application framework, DLMs are typically executed on high-performance servers with access to extensive data. This aligns with the paradigm of machine learning as a service [57] that offers data owners the capability to train and deploy their DLMs on a cloud platform without the need to construct their own computing infrastructure and development environments. In contrast, within the realm of end-edge-cloud collaboration, the DLM lifecycle activities can be carried out at the end devices, edge, or cloud servers. The end-edge-cloud collaborative computing holds significant potential in facilitating the processes of end-edge-cloud co-training, co-inference, and co-updating. For example, FL-based co-training and co-updating can be executed on end-edge-cloud nodes, while co-inference can be achieved through specific mechanisms such as early exit or model partitioning, as detailed in Section III-B.

C. Computing Power Dimension

As shown in Figure 3, cloud servers, edge servers, or end devices each offer computational support for DL. Cloud servers possess ample storage and computing resources. However, cloud-based applications face challenges such as high latency, limited situation awareness, and data privacy considerations. Edge computing, on the other hand, seeks to enhance user experience by deploying edge servers at the network edge near end devices or data sources. These edge servers process user requests with an awareness of location, delivering reduced latency and alleviating core network congestion [58]. Nevertheless, edge servers have considerably more limited computing resources than those of cloud centers and cannot efficiently host large-scale DLMs. End devices encompass a wide array of heterogeneous IoT and embedded devices, including laptops, mobile phones, industrial robots, cameras,

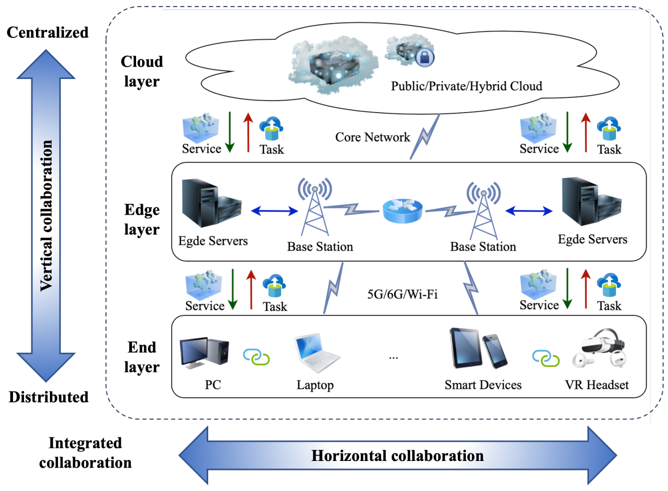

Fig. 3. End-edge-cloud computing collaboration framework.

microcontroller units, etc. While these devices can provide prompt computing services, executing DLMs, even lightweight ones, on end devices presents a challenge due to their severe computational constraints. Comparatively, end-edge-cloud collaborative computing for DL combines the advantages of these three computing modes. It facilitates the integration of data processing outcomes from different scales and locations to meet diverse application requirements.

As shown in Figure 3, collaboration modes in computing can be classified into vertical collaboration, horizontal collaboration, and integrated collaboration based on the computing hierarchy.

1) Vertical collaboration: It pertains to the cooperative and coordinated efforts spanning different layers of the end-edge-cloud computing system. Its primary objective is resource optimization and the enhancement of overall system performance by harnessing the unique capabilities of each layer.

2) Horizontal collaboration: It concentrates on the collaboration among multiple computing nodes within the same layer, such as collaboration among edge computing nodes. By horizontally distributing DL computing tasks, it can improve system scalability, robustness, and load balancing within a distributed computing environment.

3) Integrated collaboration: It represents a hybrid model encompassing elements of both vertical and horizontal collaboration. This mode involves the seamless integration and coordination of computing resources, data, and services across multiple layers and entities. It capitalizes on the strengths of each component to maximize the benefits of end-edge-cloud computing, enabling efficient and flexible DL applications.

In summary, through a comprehensive examination of the enabling factors within end-edge-cloud collaborative DL, form data, models, and computing power aspects, it becomes feasible to formulate strategies and mechanisms that efficiently leverage these elements. This, in turn, facilitates the creation and deployment of collaborative DL systems that harness the benefits of distributed end-edge-cloud computing environments as well as meet the distinct needs of various applications.

III. End-edge-cloud collaboration mechanisms for deep learning

This section is dedicated to introducing mechanisms and approaches for end-edge-cloud co-training, co-inference, and co-updating, structured around the model's lifecycle.

A. End-Edge-Cloud Collaborative Training

For a single DLM, end-edge-cloud co-training, rooted in the concept of distributed training, involves the process of training a model by utilizing multiple computational resources, including end devices, edge servers, and cloud servers. Each of these resources either holds a partition of the dataset (referred to as data parallelism) or a partition of the model (referred to as model parallelism). End-edge-cloud co-training offers several advantages: (i) It can substantially reduce the training time for large datasets and complex models that may not fit into the memory of a single machine. (ii) It leverages existing end-edge-cloud computing resources, providing cost and efficiency benefits. (iii) It has the potential to enhance overall model quality and accuracy by incorporating diverse perspectives and reducing overfitting. In the following sections, we introduce two primary parallelism approaches. Additionally, a hybrid approach that combines both data and model parallelism methods is a viable option.

1) Data parallelism-based end-edge-cloud co-training: In data parallelism-based approaches, such as FL, each computational node within the end-edge-cloud possesses a copy of the model and processes a subset of the training data. Subsequently, the updates to local model parameters are aggregated to update the global model. Data parallelism can be further categorized into centralized data parallelism and decentralized data parallelism.

- Centralized data parallelism. Parameter server architectures enable the training of DLMs with a large number of parameters on distributed clusters of machines, making it possible to process massive datasets efficiently. Typically, a central parameter server is employed to distribute and manage model parameters across multiple machines or nodes. This centralized data parallelism approach possesses characteristics such as easy deployment, high scalability, and high fault tolerance. A typical example is the federated average (FedAvg) algorithm [59]. Within the end-edge-cloud framework, a cloud or an edge computing node commonly serves as the parameter server, while each worker node typically represents an edge or end computing node. The training process under the parameter server architecture is as follows. (i) Each worker node processes its assigned data batch to calculate the gradient of the loss function with respect to the model parameters w, and then transmits the gradient information g_i to the parameter server. (ii) The parameter server aggregates these gradients g_i from all worker nodes and computes the updated model parameters for the global model using an algorithm such as $w' = w - \eta \sum_{i=1}^{N} g_i$, and distributes the global model parameters back to all worker nodes. (iii) Repeat the above steps until the global model converges.

The training process goes through multiple iterations. For parameter aggregation, the synchronous and asynchronous approaches can be adopted. In the synchronous scheme, worker nodes synchronize their model updates to a parameter server at predefined synchronization points. Participants wait for other worker nodes to complete their local training before moving to the next round. In the asynchronous scheme, participants send their model updates to a parameter server independently whenever they are ready, and do not need to wait for others. While the asynchronous scheme offers advantages in terms of flexibility and efficiency, it can also lead to parameter inconsistencies and challenges in managing the timing of updates. In contrast, the synchronous scheme ensures that each worker node starts each epoch with an identical copy of the global model. This not only theoretically assures model convergence but also ensures the reproducibility of the training process [60].

- Decentralized data parallelism. The decentralized data parallelism architecture does not rely extensively on a parameter server, utilizing it solely to initialize the global model parameters and facilitate communications among working nodes rather than participating in the model update process. Consequently, this architecture minimizes the need for frequent global information exchange. A typical training process within a decentralized structure is as follows. (i) In each epoch, a worker node updates the gradient using its local data and sends the gradient to another designated worker. (ii) The designated worker node updates the local model using its local data and the received gradients. (iii) Repeat the above steps until each worker node completes the model update, after which the final model parameters are broadcast to all participating parties.

- Personalized FL. Data parallelism is commonly employed in DL involving large training datasets or geographically dispersed data. Traditional data parallelism algorithms typically assume homogeneity among participants, with each party's training samples following an independent and identically distributed pattern. In contrast, the end-edge-cloud co-training architecture introduces variations in hardware and software configurations across computing nodes. Moreover, growing privacy concerns and data regulations such as the General Data Protection Regulation [61] have led to fragmented training data. This situation, coupled with the widespread presence of non-independently and identically distributed (Non-IID) data in real-world scenarios, will impact the performance of the global DLM. For example, the accuracy of the traditional FedAvg algorithm is notably affected by weight divergence, leading to reduced performance [62]. Furthermore, the global DLM may fail to capture individual participants' specific characteristics and preferences, since a unique global model is trained to fit different participants with heterogeneous local data distributions. To tackle these challenges, personalized FL can be integrated into the end-edge-cloud architecture. It is dedicated to solving Non-IID problems and training a personalized model for each participant while still leveraging the advantages brought by FL.

One general personalized FL approach is to train a global model and then personalize it for each participant through additional training or fine-tuning on each local dataset. This approach's efficiency depends on the performance of the global model, so many PFL methods aim to address the performance issues of global models under Non-IID datasets, such as parameter divergence, data distribution biases, and unguaranteed convergence, to improve the performance of subsequent personalization on local data.

A straightforward method for mitigating data heterogeneity among different participants involves sharing a small portion of private data or private statistical information [62]–[64]. For example, Huang et al. [63] proposed a data-sharing strategy to optimize the training process on Non-IID data by creating a small subset of data that is globally shared between all the edge nodes. Experimental results show that accuracy can be improved by 30% for the CIFAR-10 dataset with just 5% of data shared globally. However, this method may be impractical, as sending local data to the server violates the fundamental privacy assumption of FL. While privacy-preserving techniques, such as differential privacy, can be employed to address privacy concerns, they may lead to performance degradation within the FL framework. Therefore, Virtual Homogeneity Learning [65] was proposed to rectify the data heterogeneity by sharing a virtual homogeneous dataset among all clients. The virtual dataset, independent of the private datasets, can be generated from the pure noise shared across clients. Its purpose is to calibrate the features of heterogeneous clients. However, implementing virtual homogeneity learning not only imposes an additional burden on network bandwidth due to the transmission of virtual data but also necessitates careful design and construction of such auxiliary data.

An alternative method focuses on fine-tuning and optimizing the FL training process to address the challenges posed by data heterogeneity without resorting to synthetic datasets. This involves adjusting training algorithms, hyperparameters, or parameter aggregation methods. FedProx [66] introduced a proximal term to the local subproblem of FedAvg, limiting the impact of variable local updates. The proximal term offers two advantages: (i) It encourages more stable local updates by constraining them to be closer to the initial (global) model. (ii) It facilitates the safe incorporation of partial updates from selected devices. Numerous other studies have also improved and optimized FedAvg from different perspectives under non-IID datasets, such as normalization and modulation of communication frequency [67], momentum updates [68] and control variables [69], and adaptive optimizers [70].

In addition, techniques such as meta-learning, transfer learning, knowledge distillation, and multi-task learning have been applied to support personalized or device-specific modeling. For example, knowledge distillation can be used to pass on knowledge obtained from a high-performance model to enhance the performance of a simpler device-specific model, while multi-task learning enables a model to learn multiple related tasks simultaneously to provide better personalized support or device-specific modeling. These methods can be combined with FL to deal with statistical data heterogeneity, such as FedMeta [71], PerFedAvg [72], personalized FL [73], FedFomo [74], FedRECON [75] and Fedavg-Reptile [76] on federated meta-learning. Further advancements in personalized FL can be found in the literature [77].

Overall, data parallelism-based end-edge-cloud co-training offers a feasible solution for training large-scale DLMs with extensive training data. However, (i) as the number of heterogeneous nodes participating in training increases, the communication overhead associated with parameter and gradient updates among nodes may become a bottleneck. (ii) Due to the varying performances of heterogeneous nodes across the end-edge-cloud spectrum, optimal batch size allocation during training becomes essential. Failing to achieve this balance may lead to an extended training time to achieve the desired level of accuracy. (iii) To maintain synchronized and consistent model parameters across various end-edge-cloud nodes, synchronization strategies are required, which can lead to reduced training efficiency and fluctuations in training speed. (iv) Debugging and troubleshooting in the end-edge-cloud distributed environment are typically more complex, as issues may involve multiple nodes and components that are geographically distributed.

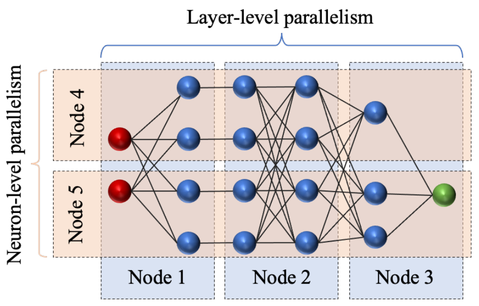

Fig. 4. Deep learning model parallelism at layer-level and neuron-level.

2) Model parallelism-based end-edge-cloud co-training: It involves the division of a DLM across multiple computing nodes in the end-edge-cloud architecture. Each node assumes responsibility for computing specific segments of the model, such as forward and backward computations. The computed results are subsequently exchanged among nodes to facilitate gradient propagation and the model parameter updates. Model parallelism strategies can be broadly categorized into layer-level parallelism and neuron-level parallelism, as shown in Figure 4. Furthermore, a hybrid approach combining both layer-level and neuron-level parallelism methods presents a viable option.

- Layer-level parallelism. Given that a DLM typically comprises consecutive layers, a straightforward model parallelism strategy involves dividing a large DLM by layers. A set of consecutive layers is grouped and assigned to a specific computing node within the end-edge-cloud architecture. These nodes perform forward and backward computations on mini-batches of data in a pipelined manner. However, a limitation of this approach is that only one node performs computing tasks at any given moment, resulting in idle periods for other nodes and, consequently, an inevitable increase in training time. To address this issue, the concept of pipeline parallelism [78], [79] has been introduced. Pipeline parallelism combines the principles of both model parallelism and data parallelism. In pipeline parallelism, each mini-batch of data is further divided into smaller batches (micro-batch). The execution is then pipelined across these micro-batches, enabling all nodes to engage in parallel computing to the greatest extent possible. Each micro-batch undergoes two passes: forward pass and backward pass. The scheduling of these passes and the aggregation of gradients vary among different approaches. Some approaches, such as GPipe [78], employ synchronous scheduling, while others, such as PipeDream [79], adopt an asynchronous approach.

- Neuron-level parallelism. It involves the partitioning of neurons, such as weight matrices, filters, and convolution kernels in a convolutional neural network (CNN), into multiple segments along a specific dimension. These segments are then processed independently on various computing nodes within the end-edge-cloud infrastructure. Megatron-LM [80] is a technique to train large-scale transformer models, in which one-dimensional tensor parallelization is implemented for the transformer's multilayer perceptron and self-attention. However, in Megatron-LM, each node is required to handle the entire set of activations, potentially causing memory bottlenecks when dealing with large-scale models. To alleviate this memory constraint, researchers have explored alternative forms of tensor parallelism, such as 2D [81], 2.5D [82] and 3D [83] parallelism, all based on the scalable universal matrix multiplication algorithm. These approaches aim to reduce memory requirements while maintaining efficient parallelization for neural network training.

Overall, model parallelism primarily serves as a solution to address the challenges in the training process, presented by extremely large-scale DLMs that surpass the capacity of a single computing node. However, (i) model parallelism introduces a notable increase in communication overhead compared to data parallelism. In data parallelism, the transfer of model parameters and gradients among different nodes is the primary communication requirement. Conversely, model parallelism necessitates the transfer of intermediate data, such as feature maps, contributing to a more substantial communication overhead. This aspect underscores the significance of addressing communication overhead as a potential bottleneck in model parallelism. Considering this challenge, a model characterized by a more locally connected structure tends to be better suited for model parallelism than one with a more fully connected architecture. (ii) Within the end-edge-cloud system, the presence of diverse computing resources, including GPUs, FPGAs, and TPUs, each possessing varying computing power and communication bandwidth, further complicates the task of optimizing the partitioning of a DLM. (iii) Not all model architectures are easily adapted to model parallelism. Some models such as recurrent neural networks may require additional engineering to make them compatible with model parallelism techniques. (iv) Pipelined model parallelism is constrained by the execution time of the front model, and any error in previous calculations can lead to a complete computational stall.

B. End-Edge-Cloud Collaborative Inference

1) Efficient end-edge-cloud progressive co-inference via early exiting: The improved performance achieved by adding more layers to a DLM comes at the expense of increased computational time and energy consumption during inference. In view of this, Teerapittayanon et al. [84] proposed a novel open-source deep neural network (DNN) framework known as BranchyNet, which incorporates early exit mechanisms through side or exit branches. As shown in Figure 5, these branches, such as Exit 1 and Exit 2, seamlessly integrate with the original baseline neural network, enabling a significant proportion of input samples to exit the network via these branches. This design can be used to meet the real-time and energy-efficiency requirements of various applications. Specifically, BranchyNet enables early exit of inference samples from the network through designated branches such as Exit 1, when specific predetermined criteria such as confidence thresholds are met. In cases where these preset requirements are not satisfied, additional layers or deeper branches such as Exit 2 or Exit 3 can be utilized to make a final determination.

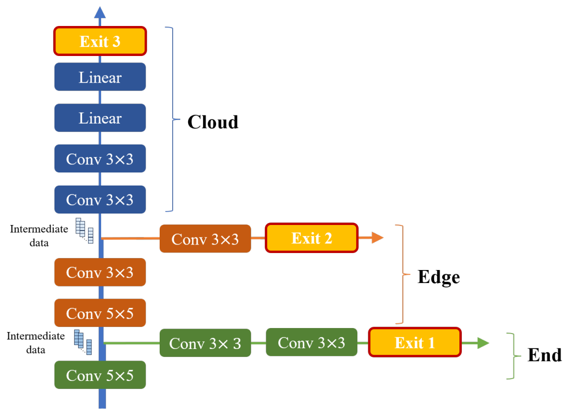

Fig. 5. BranchyNet for efficient end-edge-cloud progressive co-inference.

- End-edge-cloud progressive co-inference mechanism leveraging early exit. Early exit is a general technique and applicable to various types of DLMs, such as graph neural network (GNN) [85] and deep CNN [86]. It forms a fundamental component of the end-edge-cloud progressive co-inference, as shown in Figure 5. In this framework, a shallow DLM branch is deployed on the end (edge) node to facilitate real-time inference. When necessary (such as for fusing distributed end or edge node information or enhancing inference confidence), intermediate data (such as feature vectors generated by the end (edge)-DLM branch) can be transmitted to a deeper DLM branch residing on the edge (cloud) node for further inference. For example, Teerapittayanon et al. [87] proposed distributed DNNs over distributed end-edge-cloud computing hierarchies. It introduced end and edge DNN branches (actually some shallow portions of the DNN) with local exit points which may classify samples with high confidence locally and quickly at the end and edge devices. If necessary, the output data of the shallow DNN portions will be offloaded to the edge or the cloud for additional processing using the higher layers of the DNN.

In a broader context, the deployment of DLMs across end-edge-cloud nodes encompasses not only diverse branches of DLMs [87] but also varying scales of DLMs. Specifically, the cloud-based DLM can represent the large-scale DLM, while the DLM on the end or edge node can be achieved through model compression and lightweight techniques, such as parameter quantization [88], [89], neural network pruning [90], and knowledge distillation [7]. For example, Ren et al. [91] designed a lightweight temporal convolutional network for real-time predictions of industrial equipment's remaining useful life on the edge plane, while a large-scale temporal convolutional network on the cloud is used to provide more accurate predictions based on historical data. Yang et al. [92] proposed an end-edge collaborative fault diagnosis framework. A tiny stacked auto-encoder model is designed to operate on the end micro-controller unit for real-time decision-making, while a larger DLM is utilized on the edge node for dynamic adaptive diagnosis, thereby enhancing end inference performance. DeepAdapter [90] introduced the integration of a cloud server to build an end-edge-cloud co-inference framework for cross-platform Web applications. In the online inference process of DeepAdapter, the terminal cross-platform Web application initially conducts rapid inference using pruned CNN. If the initial inference fails to meet accuracy requirements, the intermediate data generated from the first convolutional layer is transmitted to the edge for a second inference using unpruned CNN. Additionally, when the edge experiences a high volume of concurrent requests from the terminal Web application, certain inference tasks can be offloaded to the cloud to ensure fast and real-time decision-making.

Furthermore, the combination of model compression and early exit techniques offers the potential for achieving higher compression ratios and greater flexibility in deploying DLMs on the end or edge nodes. For example, Huang et al. [88], [89] proposed an end-edge collaborative system for mobile Web applications. In this setup, a full-precision DNN is deployed on the edge server, while a binary DNN with branches is deployed on the mobile device. When the inference accuracy of all branches fails to meet the requirements, the intermediate output of the binary DNN is transmitted to the edge server for a more accurate inference, thus ensuring both inference accuracy and privacy for mobile users. Details of model compression techniques are elaborated in Section IV.

The end-edge-cloud progressive co-inference framework possesses the flexibility for both horizontal and vertical scaling. (i) Vertical scaling involves the dynamic coordination and integration of computing nodes across different layers within the end-edge-cloud architecture, including end-cloud, edge-cloud, end-edge, and end-edge-cloud collaboration. (ii) Horizontal scaling entails the addition of more end (edge) nodes dedicated to a specific task, with a subsequent fusion of their inference results using various strategies such as voting and weighting to realize a joint inference. If the joint inference outcome fails to meet predefined criteria, each node can transmit intermediate data to the upper layer for feature aggregation using feature fusion techniques such as weighting, concatenation, or tensor-product. The larger-scale DLM at the upper layer then takes the aggregated features as input, resulting in a more accurate and seamless inference.

- Exit criteria for end-edge-cloud progressive co-inference. The essence of the end-edge-cloud progressive inference mechanism lies in determining when to exit from the current computing node. Existing exit strategies include rule-based strategies and learning-based strategies.

Rule-based strategies rely on predefined rules or heuristics to determine when an early exit should occur. These rules are often made based on factors such as inference confidence, time constraints, and resource limitations. A prevalent practice is confidence threshold which involves comparing the inference confidence that is derived from a given input with a threshold denoted as t. The threshold is utilized to strike a trade-off between the latency and accuracy of the DLM. Fang et al. [86] and Laskaridis et al. [93] used the Top-1 output value from a CNN's softmax layer to estimate inference confidence. In general, higher Top-1 output values are associated with increased accuracies. However, these values can exhibit significant variations across different inputs, posing challenges in selecting an appropriate threshold.

In information theory, entropy serves as a measure of the average information or uncertainty within a random variable. Lower entropy values correspond to reduced uncertainty in a random variable such as the output vector, which can be used to measure inference confidence. BranchyNet [84] used unnormalized entropy as a metric for inference confidence, defined as follows.

$$entropy(x) = \sum_{c \in C} x_c \log x_c \tag{1}$$

where C represents the set of all possible labels, and x denotes the probability vector. DeeCap [94] calculated the entropy of the output distribution at the decoder layer as the confidence. A smaller entropy value indicates greater confidence in the inference results of the DLM. To facilitate comparisons and enhance interpretability, normalized entropy [87]–[90] is often used as follows.

$$\eta\left(x\right) = -\sum_{i=1}^{|C|} \frac{x_i \log x_i}{\log |C|} \tag{2}$$

where η has a value between 0 and 1. The entropy threshold method is commonly employed in the field of CNN-based networks. However, this method may not apply to certain tasks like regression.

The selection of the threshold value t depends on various factors, including the dataset, the specific DLM, and the application requirements. The threshold must be selected to make a trade-off between inference latency and accuracy requirements. A straightforward approach involves searching for an optimal t value across multiple randomly selected test sets [84], [86]–[89]. However, in some cases, the threshold is not static but should be dynamically adjusted according to the model's practical performance. This approach has its limitations and does not automatically yield the optimal threshold, especially in situations where end-edge-cloud communication is unstable.

In addition, early exiting rules might be set manually based on engineering experience or business requirements. For example, in real-time processing or resource-constrained applications, the decision to exit early might be made based on artificially imposed time or computational resource constraints. If the processing time or consumed resources exceed a predetermined limit, the model might be forced to exit early even if the inference confidence level has not reached the set threshold. This method is easy to implement and well-suited for (industrial) scenarios with stringent constraints.

Learning-based strategies use data-driven approaches to determine the exit points. On the one hand, possible exit points can be predicted before inference. It is feasible to use prediction algorithms, such as regression models and multi-layer perceptrons, to predict the optimal exit point based on the estimation of the input data's features. For example, Li et al. [95] proposed a predictive exit framework capable of forecasting where the network will exit, enhancing computation and energy efficiency in DL applications. On the other hand, it is possible to predict if a given sample will be correctly inferred at a specific exit point during the inference process, often by incorporating additional learn-to-exit modules in the training process. For example, Ghodrati et al. [96] used a multi-layer perceptron at each exiting point to determine whether the partially observed clip contains sufficient information to accurately classify the entire video. Xin et al. [97] proposed a learning-to-exit module, a simple one-layer fully-connected network, to predict the certainty level for a given input.

Rule-based strategies are easy to implement and can be highly effective in scenarios where the criteria for good inference are well-understood and consistent. However, rule-based strategies suffer from issues with generalization and threshold tuning, and may not always adapt well to varying data distributions or complex scenarios. For example, the confidence threshold method necessitates careful tuning of the hyperparameter t, which varies widely across different applications and datasets. Learning-based strategies are particularly relevant to pattern recognition problems. It can deal with uncertain situations such as exceptional samples and changing communication and computing environments, without the need for threshold-tuning. Consequently, it is a more promising way. However, it introduces additional computational costs, which may pose challenges for end devices with limited resources. Therefore, it is crucial to explore more efficient exit methods and strategies further. For example, Sun et al. [98] used hash functions to replace the learn-to-exit module and assign each token to a fixed exiting layer in BERT.

- Training end-edge-cloud DLMs. Like the training process for BranchyNet, a joint training method can be applied to train DLMs within the end-edge-cloud architecture. This joint training approach aims to minimize the weighted sum of the loss functions of DLMs distributed across various end-edge-cloud nodes, ensuring that each DLM or exit branch attains high accuracy commensurate with its network depth and scale [84], [99]. For example, when using the softmax cross-entropy loss function as the optimization objective, the loss function for each decision exit point that is applicable for cloud-DLM, edge-DLM, and end-DLM is defined as follows.

$$L\left(\hat{y}, y; \theta\right) = -\frac{1}{|C|} \sum_{c \in C} y_c \log \hat{y}_c \tag{3}$$

$$\hat{y} = softmax\left(z\right) = \frac{\exp\left(z\right)}{\sum_{c \in C} \exp\left(z_c\right)} \tag{4}$$

$$z = f_{exit_n}\left(x; \theta\right) \tag{5}$$

where y represents a one-hot ground-truth label vector, x denotes an input sample, with C representing the set of all possible labels. The function f_{exit_n} represents the computation of neural network layers from an entry point to the n-th exit branch, and θ denotes the network parameters, such as weights and biases for those layers. The objective during joint training is to minimize a weighted sum of the loss functions associated with each exit, as follows.

$$L\left(\hat{y}, y; \theta\right) = \sum_{n=1}^{N} w_n L(\hat{y}_{exit_n}, y; \theta) \tag{6}$$

where N represents the total number of exit points and w_n denotes the associated weight of each exit. Equal weight $w_n = 1/N$ is used in [87].

Joint training aids in learning more robust feature representations at different scale levels of the network and ensures that both the main network and the auxiliary branches are effectively trained. However, it also presents challenges including increased training complexity and resource demands, difficulty in hyperparameter tuning, and the risk of overfitting.

- Updating end-edge-cloud DLMs. To update the end-edge-cloud DLMs, various update modes can be employed, including centralized, decentralized, and end-edge-cloud collaborative approaches. (i) Centralized update involves the retraining of the entire DLM or specific model components (updating only relevant parameters) using a new dataset on a central computing server. This approach effectively leverages the computational capabilities of the central server. However, it necessitates the transmission of raw data to the remote server, giving rise to concerns related to data privacy and service latency. (ii) Local update involves the retraining of end-edge-cloud DLMs locally using newly generated data. This approach allows for rapid model updates without the need for transmitting raw data, enabling personalized adaptation to specific task requirements. However, as the dataset collected at an individual end or edge node is usually small in size, local updates may introduce challenges related to generalization and model robustness. (iii) End-edge-cloud collaborative update methods employ knowledge transfer approaches to train end or edge DLMs with assistance from cloud-DLM, as described in Section III-C.

2) End-edge-cloud distributed co-inference based on model segments: The core idea of model-distributed co-inference across the end-edge-cloud infrastructure is similar to model parallelism-based co-training. It involves the segmentation and deployment of a DLM across different end-edge-cloud computing nodes and relies on the collaboration of these distributed segments for inference. This partitioning of the DLM dramatically reduces the computational resource requirements for each segment, enabling the execution of large-scale DLMs on multiple resource-constrained computing devices. The most suitable model partitioning strategy depends on several factors, including the type and internal structure of the DLM (such as layer type, feature map shape, and filter size) and the external environment, encompassing computing and network resources [100], [101]. Just like model parallelism-based co-training as shown in Figure 4, DLM partitioning methods for distributed inference can be categorized into two types: layer-level distributed co-inference and neuron-level distributed co-inference.

- Layer-level distributed co-inference. It involves dividing the DLM at the granularity of layers and distributing these segments to the various end-edge-cloud nodes for pipelined co-inference, such as JointDNN [102] for end-cloud collaboration, BBNet [103] for edge-cloud collaboration, Cogent [104] for end-edge collaboration, and DeepX [105] for end-end collaboration. A practical example is Taobao's EdgeRec recommendation system [42], where a portion of the DLM containing embedding matrices that consume a large amount of storage resources (approximately 230MB) is deployed in the cloud. Meanwhile, the remaining parts of the model without embedding layers are deployed to end mobile devices and utilize embedding features from the cloud as input for model inference. Another case is Auto-Split [12] which has been used to realize end-cloud collaborative inference for machine vision. In a vehicle recognition system, Auto-Split divides the YOLOv3 model into end-DNN and cloud-DNN components. The terminal camera runs end-DNN and uploads the output data to the cloud, which then executes cloud-DNN and a long short-term memory model for license plate recognition. Finally, the results are returned to the terminal camera. The technical details of Layer-level partition are elaborated in Section V-A.

In layer-level distributed co-inference, the incorporation of early exit or model compression techniques can further reduce model size and execution latency. For example, Boomerang [106], an on-demand cooperative DNN inference framework in the IoT environment, uses the early exit mechanism and layer-level partitioning to achieve low latency and high accuracy inference. Edgent [107], [108] is an on-demand end-edge collaboration framework. It models the delay and energy consumption of each layer within the DNN to determine the optimal partition point. Then the DNN is divided and deployed to the end device and the edge server, allowing the end-DNN to exit early based on delay and accuracy requirements.

- Neuron-level distributed co-inference. It involves the segmentation of neurons such as weight matrix, filters, and convolution kernels into distinct parts along specific dimensions and calculating them separately on different end-edge-cloud nodes, such as MoDNN [109] for end-end collaboration and DeepThings [110] for edge-edge collaboration. Depending on the dimensionality of the division, neuron-level partitioning can be further categorized into two subtypes: channel partition and space partition, as elaborated in Section V-B.

Regarding the training and updating of partitioned DLMs, while distributed training methods are available, they are often deemed uneconomical for end-edge-cloud communication systems, due to the substantial volume of parameters and frequent transmission of intermediate data during the training process. Therefore, current research mainly adopted centralized training methods, in which the DLM is trained and updated on a high-performance server before being partitioned and deployed across the end-edge-cloud nodes.

C. End-Edge-Cloud Collaborative Updating

While Section III-A primarily focuses on end-edge-cloud distributed training for a single model, this section delves into the co-updating approach of multiple DLMs within the end-edge-cloud framework, primarily accomplished through knowledge transfer technologies.

1) Motivation: In practical applications, a common challenge arises from disparities between the distribution of the training set and the test set for DL. DLMs after deployment may not be suitable for a dynamic and constantly evolving world. This necessitates that DLMs possess the capability for lifelong learning and self-adaptation to address changing environments through updates or adaptation. A straightforward approach involves retraining or incrementally training the original DLM locally. However, there may be a lack of sufficient domain data to train a high-performance model, leading to overfitting and poor generalization capabilities. For example, in bearing fault diagnosis, obtaining fault samples can be a challenging endeavor [92]. On the other hand, this approach is inefficient as training a large DLM usually requires a lot of computing resources and time. To address these issues, sharing knowledge among different DLMs to achieve lifelong learning and domain adaptation emerges as an effective strategy, due to the following advantages.

- Learning with limited data. Knowledge acquired from a source domain can be instrumental in facilitating feature extraction and learning processes in a target domain where amassing a substantial volume of training data is often cost-prohibitive, time-intensive, or at times, impractical. For example, transfer learning technology aims to leverage knowledge from related tasks to solve new tasks with little or even no additional labeled training data.

- Improving model performance. Knowledge sharing allows a model to acquire valuable guidance and experiences, such as feature representations, model structure, and layer relationships, from other domain data and models, expediting convergence and learning. This shared knowledge can enhance the learning capabilities of related models, often leading to improved model performance, including accuracy, efficiency, robustness, and generalization. For example, knowledge distillation technology can empower student DLMs to acquire insights about robustness and resilience from a teacher DLM. This equips models to better adapt to challenging scenarios such as noise, interference, and adversarial attacks.

- Effective adaptation. Knowledge learned from one DLM can be effectively repurposed to swiftly develop and tailor models for downstream or personalized domain applications, eliminating the necessity of training these models entirely from scratch. For example, pre-trained foundation models are trained on massive, diverse, and unlabeled datasets, typically through self-supervised learning, and can be fine-tuned to numerous downstream tasks, reducing training time and the computational resources required.

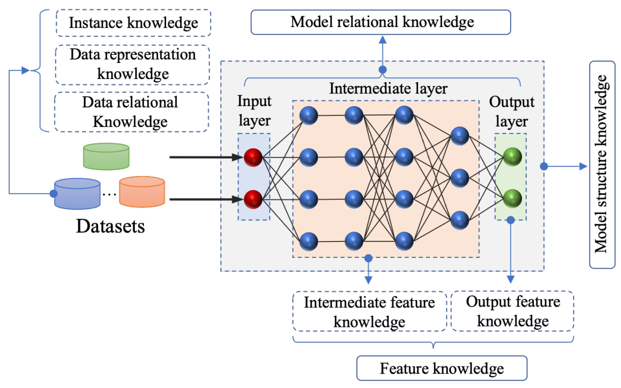
Instance knowledge
Data representation knowledge
Data relational Knowledge
Model relational knowledge
Input layer
Intermediate layer
Output layer
Model structure knowledge
Datasets
Intermediate feature knowledge
Output feature knowledge
Feature knowledge

Fig. 6. The schematic illustrations of knowledge that can be shared among end-edge-cloud nodes.

Given the compelling advantages of knowledge sharing, we integrate it into the cloud edge-end architecture to propose the concept of end-edge-cloud co-updating (learning), which will be elaborated upon in the subsequent section.

2) Categorization of end-edge-cloud co-updating: As shown in Figure 6, shared knowledge can encompass instance knowledge, data representation knowledge, data relational knowledge, and model knowledge including feature knowledge such as intermediate and output feature, model structure knowledge, and model relational knowledge. These forms of knowledge are shared through knowledge transfer techniques, such as transfer learning and knowledge distillation.

- Instance, data representation and data relational knowledge-based end-edge-cloud co-updating. These approaches involve the transfer of instance knowledge, data representation knowledge, and data relational knowledge [111] obtained from data of an end-edge-cloud node (referred to as the source domain in transfer learning, where labeled data is abundant) to train another node's DLM (known as the target domain task in transfer learning, where the labeled data is limited or the problem is different but related to the source domain). The realization of these approaches primarily relies on transfer learning techniques which will be discussed in Section VI-A, and the meaning of the above knowledge is as follows. (i) Instance knowledge assumes that certain labeled data from the source domain can be reused, such as through reweighting, to aid learning in the target domain. (ii) Data representation knowledge pertains to the common feature subspace shared by data in both the source and target domains. In transfer learning, the original features are transformed to obtain a common feature representation. (iii) Data relational knowledge involves the relations and rules among the data in the source domain. For example, techniques like angular and distance relationships [112], similarity and correlation [113], and similarity-preserving [114] are utilized to assess relationships between sample data during the training process.

- Feature knowledge-based end-edge-cloud co-updating. Feature knowledge comprises abstract features, including output features and intermediate features, acquired by DLMs processing sample data. By emulating the feature knowledge of other DLMs (referred to as the teacher model in knowledge distillation) on end-edge-cloud nodes, the local DLM (termed the student model in knowledge distillation) can be trained and updated, serving the purposes of model compression or performance enhancement.

(i) Output feature knowledge relates to the data derived from the last hidden layer or the output layer of the DLM, such as logical units before a softmax activation layer in classification tasks [111], or soft targets which obtained from the softmax layer represent class probabilities. In the extreme case, the inference result of the model can also be considered as the output feature knowledge. For example, in study [43], CloudCNN located on the cloud is used to predict labels of the uploaded data from end devices and transmit labels to the edge servers. Thus, EdgeCNN can be retrained with those labeled data, which effectively complements the edge training dataset. Output feature knowledge is widely used in the knowledge distillation process. However, the output feature knowledge from the teacher model primarily guides the training of the deep layers of the student model, rather than the shallow layer responsible for feature extraction [115]. The information capacity of hidden layers in the complex teacher model significantly differs from that of the simpler student model, leading to varying feature representation capabilities. Consequently, output feature knowledge alone may not suffice to guide the training of the student model.

(ii) Intermediate feature knowledge relates to the features extracted from the hidden layers of the teacher model and can serve as valuable hints for the corresponding hidden layers in the student model, prompting the student model to generate similar intermediate representations. For example, FitNets [116] enabled the training of a thinner yet deeper student model by leveraging the outputs and intermediate representations learned by the teacher model, to enhance the performance of the student model. Studies [117], [118] demonstrated the value of the student model learning critical intermediate features from the teacher model.

- Model relational knowledge based end-edge-cloud co-updating. Model relational knowledge involves logical relations and rules between model layers or weight parameters. Various techniques are employed to measure these relations, such as the flow of solution process matrix [119], Jacobian matrix [120], graph-based knowledge [121] and information flow [122] for measuring the relation between DLM layers. These model relational knowledge can guide other models for training.

- Structured knowledge-based end-edge-cloud co-updating. Structured knowledge refers to structured and standardized information that can be easily documented and transferred, typically involving parameters or prior distributions of DLM hyperparameters such as learning rates. For example, specific layers of the Cloud-DLM can be directly incorporated into the End-DLMs and Edge-DLMs, as shown in Figure 7. There are 5 and 3 convolutional layers in the Edge-DLM and End-DLM originating from the Cloud-DLM. In the Edge-DLM and End-DLM training processes, these shared layers are frozen with only the remaining layers being updated to achieve fast convergence. Furthermore, domain adaptation of the pre-trained foundation model to downstream tasks is another typical example of structural knowledge transfer, which can be realized through fine-tuning, prompt tuning, instruction tuning, etc.

In practical applications, the sharing of structured knowledge in the end-edge-cloud environment has been widely studied. For example, Lu et al. [123] proposed a model parameter-sharing approach in their edge-cloud collaborative learning method, COLLA, for user behavior prediction. Here, model parameters from a cloud-trained model which uses historical data, are distributed to edge devices. Each edge device conducts incremental learning using local data to build a personalized edge model. The cloud model acts as an aggregator, consolidating knowledge (model parameters) from multiple edge models and sharing these parameters with edge devices to alleviate overfitting issues arising from limited local data. Ding et al. [43] proposed an edge-cloud collaboration framework for cognitive services. They deployed a shallow model (EdgeCNN) on the edge server to deliver prompt cognitive services and a deep model (CloudCNN) on the cloud server to enhance the performance of the EdgeCNN by sharing predicted labels, model structure and parameters. The experimental results showed that EdgeCNN reduced the average response time of cognitive services by up to 55.08% and improved accuracy by up to 26.70% within the edge-cloud collaboration framework. Jing et al. [124] designed an edge-cloud collaboration framework for predicting the remaining useful life of machinery. In this framework, the cloud prediction engine (Cloud-PE) with a built-in deep prediction DLM is encapsulated in the cloud service layer, while the edge prediction engine (Edge-PE) equipped with a shallow prediction DLM is positioned in the edge service layer. Cloud-PE plays a pivotal role in assisting Edge-PE to achieve fast and highly accurate predictions by providing soft supervision and sharing depth model parameters. Experimental results show that the edge-cloud collaboration framework yields more accurate predictions and reduces the training time of the edge model compared to existing data-driven prediction methods.

3) Federated continual learning: FL involves distributed training across multiple heterogeneous devices without sharing private local data. It is a commonly used knowledge transfer strategy in the end-edge-cloud architecture. Standard FL, even personalized FL follows a static configuration, where the local data and task don't change over time. However, in realistic tasks, it is common to collect data progressively while the training goes on, and can be required to learn new tasks over time. Recently, some interesting studies have been devoted to exploring federated continual learning (FCL) which assumes that the learned tasks change over time, where multiple clients $C_c \in \{C_1, C_2, \cdots, C_c\}$ are continuously trained on a sequence of tasks $\{T_c^1, T_c^2, \cdots, T_c^t\}$ from the private data stream while communicating the learned parameters or knowledge with a global model.

On the one hand, some FCL methods make use of the parameter isolation strategy and try to distinguish old knowledge from new knowledge from the perspective of parameters. FedWeIT [125] decomposed the network weights into global federated parameters and task-specific parameters which include local base parameters and task-adaptive parameters. Each client $C_c \in \{C_1, C_2, \cdots, C_c\}$ receives selective knowledge from other clients by taking a weighted combination of their task-specific parameters. In FedWeIT, a set of the model parameters $\theta_c^{(t)}$ for task t at client c is defined as $\theta_c^{(t)} = B_c^{(t)} \odot m_c^{(t)} + A_c^{(t)} + \sum_{i \in C_{\setminus c}} \sum_{j<|t|} \alpha_{i,j}^{(t)} A_i^{(j)}$, where $B_c^{(t)}$is the set of base parameters for client c shared across all tasks in the client, $m_c^{(t)}$is the set of sparse vector masks which allows to adaptively transform $B_c^{(t)}$ for the task t, $A_c^{(t)}$is the set of a sparse task-adaptive parameters at client c, $\alpha_{i,j}^{(t)}$ is the weight. Dong et al. [126] developed a global-local forgetting compensation model to alleviate the catastrophic forgetting in FL, which employs a class-aware gradient compensated loss and a class-semantic linkage distillation loss to equalize the gradient propagation and guarantee stable inter-class relations across tasks. Le et al. [127] presented an FCL scheme based on broad learning, where a weighted processing strategy is proposed to solve the catastrophic forgetting problem and a local-independent training solution is proposed to support fast and accurate training.

On the other hand, some studies try to conduct FCL from the perspective of knowledge distillation and extraction. For example, Ma et al. [128] proposed continual FL with distillation, which performs knowledge distillation on both the clients and the server, with each party independently having an unlabeled surrogate dataset and different learning objectives such as learning the new task and reviewing old tasks. FedKnow [129] is a client-side FCL framework composed of a knowledge extractor, a gradient restorer, and a gradient integrator that combines signature tasks identified from past local tasks and other clients' current tasks through the global model.

However, these approaches ignore the maintenance or consolidation of old knowledge, which will inevitably lead to a degradation in the performance of the model on previous tasks due to the probabilistic bias problem [130]. On the other hand, these approaches ignore asynchronous learning [131] where the continual learning of multiple tasks happens at each client with different orderings and in asynchronous time slots. The asynchronous nature of each client's learning can lead to an imbalance in parameter variations. Although Wang et al. [130] proposed a federated probability memory recall framework to mitigate the probability bias problem and the imbalance in parameter variations, and Shenaj et al. [131] considered an asynchronous FL setting, there is still a large space to improve for existing FCL approaches. Meanwhile, efficiency and communication overhead are also the key issues of FCL. In addition to data replay strategies, efficient and secure communication strategies need to be further explored. For example, Zhang et al. [132] proposed the bidirectional compression and error compensation algorithm to produce the communication-efficient FCL method.

D. Summary and Lessons Learned

In this section, we derive a holistic taxonomy for the end-edge-cloud collaborative learning, including end-edge-cloud co-training, co-inference, and co-updating throughout the entire DL lifecycle. The summary and lessons learned are as follows.

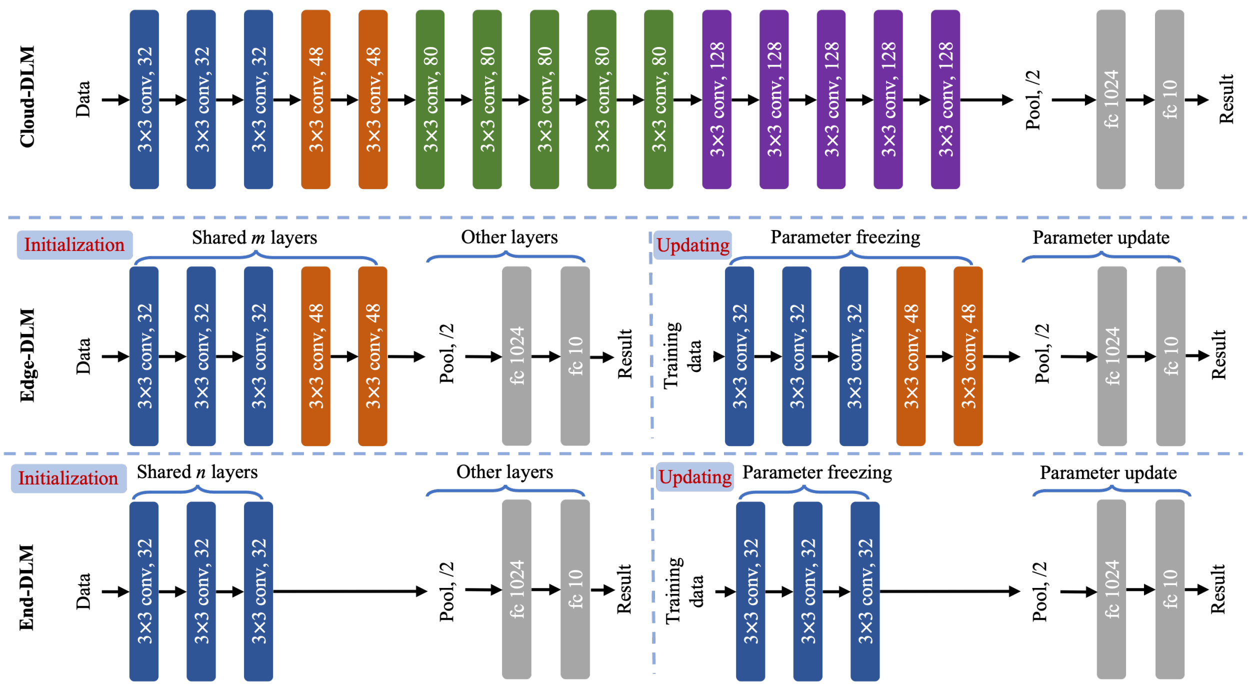

Fig. 7. Structure knowledge sharing between Cloud-DLM and End-/Edge-DLM.

1) Heterogeneous end-edge-cloud co-training: End-edge-cloud co-training draws inspiration from traditional distributed training, particularly FL. However, it grapples with a more intricate computing environment. This complexity arises from several sources: (i) Heterogeneous end-edge-cloud computing devices, which exhibit heterogeneity in storage, computing power, and communication capabilities even within the same end, edge, or cloud layer. (ii) Heterogeneous Data, characterized by differences in quantity, quality, and distribution across computing nodes, especially Non-IID data. (iii) Heterogeneous models, each tailored to specific application scenarios on end-edge-cloud nodes. To address the challenge of device heterogeneity, it is crucial to optimize both the architectural and mechanistic aspects of data parallelism and model parallelism. When dealing with data and model heterogeneity, strategies such as personalized FL, user context, transfer learning, meta-learning, knowledge distillation, and multi-task learning could be considered.

2) Dynamic end-edge-cloud co-inference: Co-inference, unlike co-training, does not entail backward computation, which simplifies its deployment in practical applications. However, the heterogeneity and dynamics of the end-edge-cloud devices and networks remain significant challenges. Selecting the optimal co-inference framework for each service quality can be a daunting task. Therefore, on the one hand, the performance and service improvements brought by end-edge-cloud co-inference, such as delay, energy consumption and robustness should be quantitatively evaluated, and the trade-offs between different indicators should be carefully considered. On the other hand, advanced co-inference mechanisms, such as early exiting and model partitioning and cooperation, should be explored to cope with complex end-edge-cloud environments.

3) Knowledge-based end-edge-cloud co-updating: In a dynamic application environment, DLMs need continuous updating to achieve lifelong learning and personalization. To alleviate the challenges associated with model updating, knowledge-sharing within the end-edge-cloud framework emerges as a valuable approach. However, due to interpretability issues in DL, what knowledge is reasonable and efficient to share is not always supported by theoretical analysis. Moreover, heterogeneity in data, devices, and models complicates knowledge-sharing mechanisms. Therefore, on the one hand, we need to quantify performance improvements from knowledge sharing and strengthen the exploration of interpretability theory in DL. On the other hand, we still need to explore the knowledge-sharing mechanism of end-edge-cloud co-updating to adapt to heterogeneous environments.

4) Future trend: Advanced pre-trained foundation models such as ChatGPT and PanGu have gained significant attention due to their unprecedented ability to capture rich knowledge from large-scale pretraining data, which gives large models the potential to serve as universal experts or domain experts through fine-tuning. However, large-scale deployment of foundation models still faces challenges including personalization, computing power, and efficiency. To address these challenges, one promising deployment scheme is large and small model collaboration in end-edge-cloud computing power network (CPN). On the one hand, the synergy between these two distinct DL paradigms results in a more responsive, efficient, and adaptable DL ecosystem. On the other hand, the CPN that converges end-edge-cloud computing and networks can flexibly allocate computing resources on demand to meet the anytime and anywhere computing requirements of DL systems. Therefore, the convergence of large and small model collaboration and CPN is a novel and inevitable paradigm after the convergence of DL and end-edge-cloud computing.

IV. Model compression technologies for deep learning

End-edge-cloud collaborative learning necessitates the deployment of a complete or at least a portion of the DLM on end and edge nodes that are close to the data source. However, the inherent limitation of computational resources on these end and edge nodes calls for the adoption of model compression techniques. Popular DL model compression techniques include pruning and sparsification, parameter sharing and quantization, manual design of lightweight networks, and neural network architecture search.

A. Pruning and Sparsification of Neural Networks

A common method for compressing neural networks is weight pruning (unstructured pruning), which aims to remove "redundant" weight parameters in the network. However, this often leads to irregular, sparse weight matrices, which rely on indices for storage, making them less compatible with the data parallel execution model by GPUs and multicore CPUs. To overcome such limitations, recent studies proposed the idea of structured pruning, which incorporates regularity or structures in weight pruning, i.e., weight matrix's row pruning, filter pruning, and self-attention matrices pruning. Structured pruning, while coarser in granularity and still containing some redundant weights, maintains a full matrix with reduced dimensions, making it more suitable for general-purpose GPUs or multicore CPUs. Whether employing structured or unstructured pruning, the ultimate goal is to remove the defined unimportant portions of the network. According to the pruning operation stage, existing pruning strategies can be divided into pruning at initialization and pruning after training.

1) - Pruning after training: Pruning after training follows the training-pruning-retraining (fine-tuning) methodology. Initially, the DLM is trained, after which the importance of parameters or nodes in the neural network is evaluated, and redundant ones are pruned. Subsequently, the network is retrained or fine-tuned. The above process is iterated until the model converges. This approach is widely used in existing pruning algorithms, as detailed in recent review literature [133]–[136], However, pruning-after-training approaches are less efficient in terms of learning, as the iterative pruning and retraining (fine-tuning) processes consume considerable time and computational resources to obtain the desired DLM.

2) - Pruning at initialization: In contrast to pruning after training, pruning at initialization aims to sparsify neural networks during the random initialization stage. A traditional way to obtain sparse DLMs involves incorporating sparse constraints during DLM training, such as using sparse representation, sparse cost functions, and sparse regularization [137]. While this method has the advantage of not requiring extensive time and computational resources for pre-training the model, it typically results in low sparsification rates and the neural network cannot be compressed significantly. The recent lottery hypothesis [138], [139] challenges the belief that large neural networks are necessary for achieving high performance. This hypothesis suggests the existence of a sparse network within a larger neural network, with comparable performance to the original network. The concept of "mask" is introduced to identify this winning network. The steps are as follows.

Step 1: Randomly initialize the parameters of the original neural network to obtain $f\left(x, w_0\right)$, where x represents the input data and w_0 is the network parameter;

Step 2: Train the original neural network j times to acquire w_j;

Step 3: Set a corresponding binary mask m for each parameter, with values of 1 and 0;

Step 4: Set a threshold value based on the desired pruning ratio. Assign a value of 0 to parameters whose absolute value is less than the threshold and set their corresponding masks to 0. Reset parameters with absolute values greater than the threshold to the parameter value w_0 (before training) and assign a mask value of 1.

Following these steps, the sparsely initialized network $f\left(x, m \odot w_0\right)$ can be obtained. Various methods or algorithms have been developed to identify sparse networks within randomly initialized neural networks, including supermask [140] and edge-popup algorithm [141], which are detailed in the review literature [142]. It is worth noting that most studies focus on unstructured pruning rather than structured pruning.

Overall, neural network pruning after training typically removes redundant parameters or neurons from a trained model, while pruning at initialization aims to prune a randomly initialized model. Although pruning at initialization is an interesting idea, it is inferior to the pruning after training method in terms of practical performance. On the one hand, it does not bring much change to the pruning ratio, criterion, and schedule [142], for example, many pruning criteria used in pruning-at-initialization approaches resemble those in pruning-after-training approaches. This results in the same, or even worse performance of the final pruned models, even with different pruning methods. On the other hand, most pruning at initialization studies focus on unstructured pruning rather than structured pruning, leading to the inability to obtain efficient training acceleration due to the lack of sparse training library support. Furthermore, dynamic-mask approaches exacerbate this issue compared to static-mask approaches, as their variable masks hinder hardware acceleration.

B. Parameter Sharing and Quantization of Neural Networks

Parameter sharing and quantization represent an important compression method, aimed at efficiently mapping DLM parameters to a smaller amount of data. Common methods include parameter clustering, hash sharing, parameter quantization, etc.

1) Parameter clustering: This method achieves parameter sharing by clustering methods such as k-means clustering. It assigns the same index to similar parameters within a pre-trained DLM and uses specific statistic indicators like the mean value, of the index to represent this type of parameter [143], [144].

2) Hash sharing: It uses hash functions to facilitate parameter sharing. For example, Chen et al. [145] introduced a low-cost hash function that groups weights into hash buckets to enable parameter sharing. One advantage of this approach is that it does not necessitate pre-training the DLM. Moreover, the shared parameter vector can be initialized in the same way as the normal model parameter initialization, and the DLM based on hash-sharing can be trained conventionally.

3) Parameter quantization: It entails reducing the precision of the floating-point representation used for the parameters (weights and activations) of the network to conserve storage and computation resources, for example, converting 32-bit floating point into low-precision data types such as 8-bit or 4-bit integer. In typical cases such as binary neural networks [146], [147], floating-point weights are replaced with binary representations. As a result, a model that originally used 32-bit floating-point representation can be compressed by a factor of 32 to suit resource-constrained end devices and edge nodes. However, parameter quantization causes information loss and introduces discontinuities that can complicate DLM optimization, leading to a significant drop in model accuracy [148]. Therefore, the trade-off between data bit width and model accuracy is a crucial issue. Quantization bit width was traditionally determined through manual experimentation [149], [150]. Khoram et al. [151] proposed an adaptive method for determining the quantization bit width of each parameter based on the gradient of the corresponding loss function. Several studies [152]–[154] have emerged to support the adaptive determination of the optimal quantization bit width based on the specific task environment. Moreover, to minimize accuracy loss due to quantization, Yu et al. [155] introduced compression methods that use convolutional kernels as quantization units. Zhou et al. [156] proposed incremental quantization methods that map weights of trained CNNs to bits in the form of exponential powers of 2, facilitating model computation through binary shift operations.

Overall, (i) parameter clustering groups similar parameters together, which can significantly reduce the model size. However, it may lead to a loss of fine-grained information, potentially affecting model performance. (ii) Hash sharing uses a hash function to group parameters. This approach can efficiently reduce memory requirements but might introduce hash collisions, where different parameters are erroneously mapped to the same representation, leading to potential performance degradation. (iii) Parameter quantization reduces the precision of the parameters, converting them from floating-point to lower-bit representations. This method dramatically decreases model size and can speed up inference by leveraging integer arithmetic. However, aggressive quantization can lead to a significant loss in model accuracy due to the reduced precision. Therefore, they must be carefully implemented to balance the trade-off between model size, inference speed, and accuracy.

C. Manual Design of Lightweight Networks and Neural Architecture Search

1) Manual design of lightweight networks: Some studies focus on crafting efficient lightweight modules and networks also with the purpose of network optimization, such as the fire module [157], Xception module [158], depthwise convolution module [19], inverted residual with linear bottleneck [159], and squeeze-and-excitation module [160]. Designing lightweight networks manually necessitates considering various factors such as inter-layer connections, DNN depth, and the computational setting of convolutional layers, the need to continuously adjust network structures according to model performance. Effective design relies on extensive designers' expertise, domain knowledge, experience, and intuition. However, experience and intuition-based manual design does not guarantee optimality for the target task, and it is impractical to manually search the large solution space for the optimal one.

2) Neural architecture search: In contrast, neural architecture search (NAS) [161] similar to hyper-parameter optimization in machine learning, automates the exploration and design of neural network architectures to achieve peak performance for a specific task. NAS techniques consist of three main components: the search space, search strategy, and performance evaluation. NAS-based lightweight network design aims to obtain lightweight network structures through NAS technology.

- Indirect search. One approach is to use existing lightweight neural network structures as a framework and then fine-tune the network using NAS. For example, ProxylessNAS [162], MDENAS [163], MNasNet [164], MobileNet-V3 [160], FB-Net [165], ChamNet [166], EfficientNet [167] and FB-Net-V2 [168] all build upon the MobileNetV2 as the basis for their search and adjustments. The search space for MobileNet cells mainly includes the type of separable convolution, the type of skip connections, channel count, convolutional kernel size, expansion ratio, and layer count.

- Direct search. An alternative method involves directly searching for lightweight network structures. In [169], binarized NAS, with a search space consisting of binarized convolutions, is introduced to partially connected differentiable architecture search to produce highly compressed models. Chen et al. [170] introduced channel sampling and operation space reduction into a differentiable NAS to significantly reduce the cost of searching. They used a performance-based strategy to discard less effective potential operations. Shen et al. [171] presented an automatic search framework for compact and accurate binary CNN, encoding the number of channels in each layer into the search space and optimizing it using evolutionary algorithms. Experiments showed that this method can yield binary CNN with acceptable model sizes and computational overheads, achieving performance matching that of full-precision models.

- Search strategy. Various search strategies are available, including random search, Bayesian optimization, evolutionary algorithms, reinforcement learning, and gradient-based methods. Recent NAS research focused on exploring the search space through reinforcement learning and gradient-based methods [172], [173].

Overall, NAS has tremendous potential in automating the design of high-performance lightweight networks. However, it has several limitations as follows. (i) In indirect search method, the utilization of pre-existing lightweight neural network structures/frameworks for subsequent fine-tuning via

NAS strategies tends to constrain the exploration space to predefined architectures. This approach may inadvertently overlook more optimized or innovative network structures. (ii) In direct search method, although it allows for direct exploration of lightweight network structures, setting up the search space itself can be challenging. A search space that is too large can lead to inefficient searching, while one that is too small might not contain effective network architectures. (iii) NAS typically requires substantial computational resources. To find the optimal network architecture, it is necessary to evaluate a large number of different architectures, which often involves fully training each architecture and assessing its performance. Although some recent methods aim to reduce this computational burden, NAS remains a resource-intensive process. (iv) NAS involves multiple components and choices of hyperparameters, resulting in architectures that lack explainability, making them hard to understand and adjust.

D. Summary and Lessons Learned

The deployment of DL on IoT and embedded devices is a crucial aspect of the end-edge-cloud collaboration framework. In this section, we review the recent mainstream DLM compression approaches and derive the following lessons.

1) Holistic and systematic considerations for pruning and sparsification: Pruning methods come in two primary forms: pruning after training and pruning at initialization. Pruning after training involves pre-training a large over-parameterized model and then iteratively pruning unimportant weights, followed by fine-tuning. Pruning at initialization prioritizes model structure overweights and aims to create a sparser network from the start. However, model structure and parameters are interconnected, necessitating the development of holistic and systematic pruning methods that strike a balance between the two aspects. This also holds promise for enhancing the interpretability of DLMs.

2) Parameter sharing and quantization require collaboration with other methods: Parameter sharing involves mapping parameters to a smaller data space, effectively reducing storage requirements, particularly in fully connected layers. However, it is not easy to generalize, for example, its application to convolutional layers remains challenging. Relatively, parameter quantization has good compatibility and can be combined with other compression techniques in practical use. However, it faces limitations in terms of compression rates, potentially compromising network capacity and feature quality. In addition, quantization introduces noise into gradient information, making convergence in gradient-based training processes more challenging, with binarized networks experiencing more significant accuracy drops.

3) Manual design and NAS demand creativity and exploratory: On the one hand, manual design of lightweight networks relies on designers' expertise, intuition, and experimentation. These networks often have limited compatibility with other compression and acceleration methods due to their unique structures. Meanwhile, models based on lightweight convolutional kernels may struggle to generalize features effectively due to their restricted capacity. On the other hand, current NAS methods are typically built upon existing models as the backbone network and involve the manual selection or heuristic strategies for structure search, limiting the search space and potentially leading to suboptimal solutions. Future developments may involve leveraging strategies such as reinforcement learning to automate network structure searches and achieve better-performing network structures.

4) Future trend: The rapid advancement of large-scale foundation models exemplified by ChatGPT and Pangu has ushered in a demand for efficient and cost-effective deployment on the edge or end devices. However, existing compression and acceleration techniques are primarily tailored for CNN, and often focus solely on software-level optimizations. Consequently, it has become an imperative trend to craft dedicated compression techniques and hardware architectures explicitly tailored to large models.

V. Model partitioning technologies for deep learning

Model partitioning is a crucial aspect of both end-edge-cloud model-distributed co-training and co-inference. Compared with inference that involves only forward computation, parallel training in the context of end-edge-cloud necessitates frequent exchanges of parameter and gradient information between nodes. This can lead to reductions in both generalizability and efficiency. Consequently, both industry and academia have placed significant focus on model-distributed co-inference. This section aims to introduce model partitioning techniques specifically designed for end-edge-cloud co-inference. Relevant concepts and techniques presented here can also offer valuable insights for co-training scenarios.

A. Layer-level Partitioning

The concept of layer-level partitioning for end-edge-cloud co-inference stems from several key observations as follows. (i) The overall execution time of a DLM is influenced by both model computation time and data transfer time. In cloud-based DL, the primary bottleneck lies in the data transfer overhead between data sources and the cloud. (ii) There are intermediate layers within DLMs where the output data volume is significantly smaller than the original data, presenting an opportunity for lower latency and reduced energy consumption during data transmission. For example, the input data volume of tiny YOLOv2 is 0.95 MB, whereas the output data volume of the intermediate layer (max5) is only 0.08 MB, representing a reduction of 93% [174]. (iii) Enhanced computing power can reduce the computation time of a DLM or its components, potentially resulting in time savings that outweigh the data transmission time between end-edge-cloud nodes.

These observations form the basis for dividing a DLM into multiple parts and deploying them across end-edge-cloud computing nodes. As shown in Figure 4, the segments run serially in a pipelined manner, with each partition's output serving as the input for the next, all interconnected through a network. A key challenge in implementing this layer-level computation partitioning strategy is determining the optimal partition point within a DLM while considering resource constraints, such as energy, computing power, storage, and dynamic network conditions. The goal is to meet performance requirements such as latency and accuracy.

One notable attempt to address this challenge is Neurosurgeon [101]. First, it predicts the execution time and energy consumption of each layer on both end and cloud nodes using specific regression models based on DLM types. These prediction models utilize input variables like the size and number of input and output feature maps, and the number of input neurons. Then, Neurosurgeon evaluates various partition schemes to identify the optimal partition point that minimizes latency or energy consumption for end-edge-cloud scenarios. In experimental tests of 8 DLMs, Neurosurgeon achieved an average runtime acceleration of 3.1 times (up to 40.7 times) and an average energy consumption reduction of 59.5% (up to 94.7%) for mobile devices when compared to cloud-only DLM execution. Moreover, the framework increased cloud data throughput by an average of 1.5 times (up to 6.7 times).

Neurosurgeon has opened up the research field of layer-level computation partitioning for end-edge-cloud co-inference. However, it still has certain limitations that warrant consideration.

1) Optimality: Neurosurgeon aims to divide a DLM into two parts and execute the front part on the end device and the back part on the cloud server. It may not be the optimal solution, because multiple partition points can exist within a DLM, allowing various layers to be computed on different end-edge-cloud nodes. Moreover, the prediction models used by Neurosurgeon to estimate latency and computing power consumption on mobile devices and cloud servers suffer from prediction accuracy issues. This is exacerbated by the fact that runtime optimization by programming frameworks like Caffe, TensorFlow, and PyTorch causes the execution time of a sequence of layers to differ from the sum of independent execution times for each layer [102], [175].

2) Adaptability: As DLMs become more complex, incorporating features like residual connections, dense connections, and attention mechanisms, they often adopt directed graph structures, rather than simple chain-like ones, for example, GoogleNet and ResNet with directed acyclic graphs (DAGs), long short term memory or recurrent neural networks (RNNs) with recurrent structures. However, Neurosurgeon and subsequent studies [176]–[178] are better suited for simpler, chain-like DLMs such as Tiny-YOLOv2, VGG16, and AlexNet, and can not optimally partition DLMs with more complex structures like DAGs or recurrent structures [174]. To handle such complex DLMs, more advanced graph-theory analysis is required.

3) Dynamicity: Neurosurgeon's prediction models are constructed in advance for specific hardware platforms and network environments. However, the increasing diversity of end-edge-cloud devices and the volatile nature of network conditions pose challenges. Creating prediction models for each unique hardware and network environment would require extensive measurements and resources, making it impractical to adapt to dynamic changes in the end-edge-cloud environment.

In light of the above issues, Hu et al. [174] researched the layer-level partition of DLMs characterized by DAGs. They proposed a dynamic adaptive DNN surgery (DADS) scheme and formulated the partition problem as a min-cut problem on a DAG under the condition of light network loads. As shown in Figure 8, for a neural network M, DADS constructed a DAG model $G = \langle V, E \rangle$, where $V = \{v_1, v_2, \ldots v_n\}$ denotes the n vertices in G, with each $v_i \in V$ corresponding to a layer in M, v_1,v_n represent the input and output layers respectively, while edges depict dependencies between layers v_i, v_j. Based on G and subsequent steps, a latency graph G' as shown in Figure 9 can be built, paving the way for formulating the optimal DLM partition problem as a min-cut problem on G'.

Step 1: Add vertices e and c to G which represent the starting and ending points of the DAG, respectively. In Figure 9 (b), these vertices are highlighted in red and grey;

Step 2: Connect the vertex e to all vertices in V, with the edges' weights $< e, v_i >$ indicating the execution time of layer i in the cloud. This is illustrated by the red edges in Figure 9 (b);

Step 3: Connect all vertices in V to vertex c with the edge weight $< v_i, c >$ representing the execution delay e_i of layer i at the edge. These connections are depicted as the blue edges in Figure 9 (b).

Step 4: The weight of each edge $< v_i, v_j >$ in E is set to the data transfer delay t_i from vertex v_i to vertex v_j. If a vertex v_i has more than two succeeding vertices, an additional virtual vertex $v_i{}'$ is added after v_i, and the edge weights $< v_i, v_i{}' >$ and the edge weight $< v_i{}', v_j >$ are set to t_i and $+\infty$, respectively. An example is the vertex v_1 in Figure 9 (b).

Once the latency graph G' is constructed, DADS utilizes a DNN surgery light algorithm to identify a minimum link cut that divides the vertices into two disjoint sets V_e and V_c, achieving the minimum total inference latency T_{total}. The neural network layers corresponding to the vertices in V_e and the vertices in V_c are executed at the edge and in the cloud respectively. For example, $V_e = \{v_1\}$ and $V_c = \{v_2, v_3, v_4\}$ in Figure 9 (b). Meanwhile, the total inference latency is calculated as $T_{total} = T_e + T_c + T_t$, where $T_e = \sum_{v_i \in V_e} e_i$ represents the total execution latency at the edge, $T_c = \sum_{v_j \in V_c} c_j$ denotes the total execution latency in the cloud, and $T_t = \sum_{(i,j) \in E_c} t_i$ signifies the total transmission latency across the links connecting V_e to V_c.

However, as significantly shorter computation time for each layer is needed in the cloud compared to edge nodes, DADS may opt to allocate all neural network layers to the cloud. This choice is impractical and neglects the advantages of edge nodes and end devices. Furthermore, the execution of DADS can be time-consuming. For example, it took over 18.14 seconds on the Raspberry Pi 3B platform to divide the ResNet-18 model with 111 layers [175]. Therefore, Zhang et al. [175] introduced an additional transmission cost τ during the construction of the latency graph G' to enhance the realism of the partition scheme. As shown in Figure 9 (c), this modification involved adding a vertex o and three links $< e, o >$,$< o, c >$ and $< o, v_1 >$ with weights of $+\infty$, 0, and a model-dependent value τ, ensuring that $\tau > \sum_{v_i \in V} (e_j - c_j) + t_1$. A new method for measuring layer computation latency was also proposed, with $t_i = T(1, i) - max_{j \in i-1} T(1, j)$, where t_i

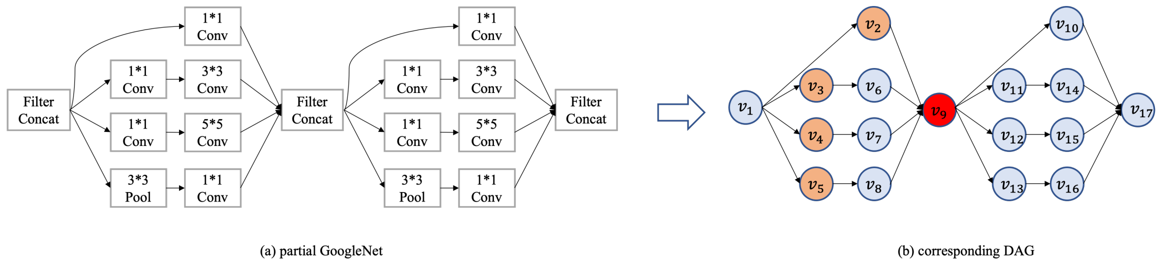

Fig. 8. Partial GoogleNet and corresponding directed acyclic graph [174].

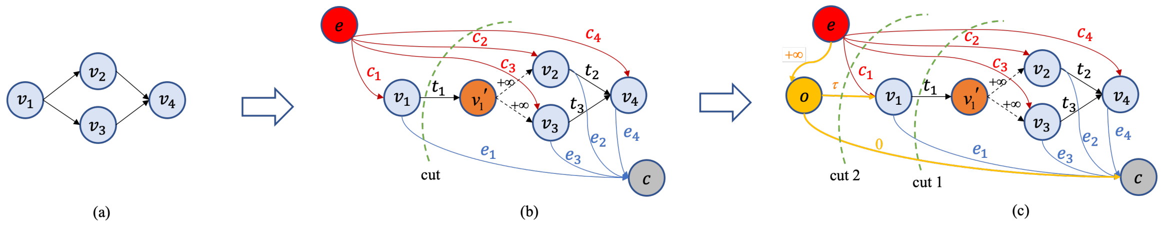

Fig. 9. Constructing the latency graph G' from G representing the DLM.

represents the execution time of the i-th network layer, and $T(1, i)$ denotes the total time required to execute the 1st to i-th network layers. Since the optimal partition of the model must be between two adjacent cut vertices in the DAG, a two-stage algorithm, called QDMP (quick deep model partition), is proposed to solve the min-cut problem on a DAG. Compared to DADS, the problem-solving speed has been improved by 66.3 times.

Although QDMP can calculate a more accurate inference time for each layer, QDMP and DADS only obtain one partition point and fail to jointly optimize the delay and energy consumption, potentially resulting in suboptimal solutions. In contrast, JointDNN [102] offers the capability to optimally segment a model into two or more parts for collaborative computation across end and cloud nodes. Moreover, it can be extended to handle complex neural networks such as those with residual connections. JointDNN introduces the concept of the JointDNN graph model, which represents a DNN as a sequence of distinct layers with a linear topology. As depicted in Figure 10, node $C_{i:j}$ represents that layers i to j are computed in the cloud, while node $M_{i:j}$ indicates that layers i to j are computed on the mobile device. Using this representation, JointDNN transforms the problem of partitioning DNN computations into finding the shortest path from S to F in a graph. For both training and inference, it constructs an integer linear programming model by incorporating constraints such as mobile battery limitations, cloud congestion, and quality of service requirements. This model aims to find the optimal partition of the model with minimal latency or energy costs. However, as JointDNN relies on an application-specific profiling method to measure the runtime DNN architecture, network execution latency, and energy consumption, it may not be effective in dynamic end-edge-cloud scenarios.

Overall, DADS, QDMP, and JointDNN possess the capability to model DLMs as DAGs and perform partitioning. Despite their respective advantages, they still face common issues and challenges as follows. (i) These approaches require prior measurements of computation latency for network layers specific to certain hardware and communication environments. This lack of adaptability to dynamic environments can be a limitation. (ii) All three approaches rely on maintaining a complete DLM on both the end (edge) and the cloud to support dynamic partitioning. However, as DLMs continue to grow in scale, storage space on end or edge nodes becomes a constraint. Combining model compression techniques can be a potential solution to further reduce model size and computation load. For example, Auto-Split [12] jointly optimized the split point and bit-widths for weights and activation of layers on the edge device, while BBNet [103] combined channel-pruning, feature compression, and model partition methods. (iii) Network communication remains a bottleneck for end-edge-cloud scenarios. To address this challenge, data compression techniques, such as lossless compression [102], lossy compression [176], and JPEG feature encoder [179], can be used to reduce the amount of data transmitted among end devices, edge nodes, and clouds. However, it's important to note that data compression often comes at the cost of a certain level of inference accuracy.

TABLE IV
PARAMETER DEFINITION OF JOINTDNN GRAPH MODEL

Param	Description
U_1	The cost of uploading the input of the first layer
$ME_{i:j}$	The cost of executing layers i to j on the mobile
$CE_{i:j}$	The cost of executing layers i to j on the cloud
$EU_{i:j}$	The transition cost from the mobile to the cloud
$ED_{i:j}$	The transition cost from the cloud to the mobile
ϕ_k	All the following edges:$\forall i = 1 : k-1\ ED_{i,k-1}$
Ω_k	All the following edges:$\forall i = 1 : k-1\ ME_{i,k-1}$
Ψ_k	All the following edges:$\forall i = 1 : k-1\ EU_{i,k-1}$
Γ_k	All the following edges:$\forall i = 1 : k-1\ CE_{i,k-1}$
Π_m	All the following edges:$\forall i = 1 : n\ ME_{i,n}$
Π_n	All the following edges:$\forall i = 1 : n\ ED_{i,n}$

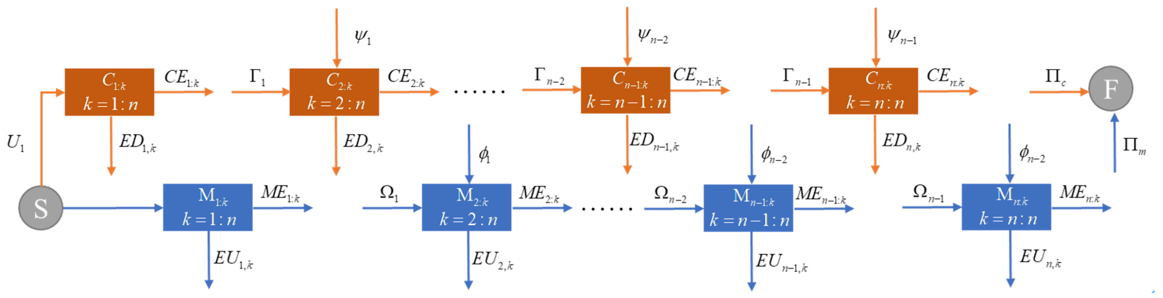
Fig. 10. JointDNN graph model. The shortest path from S to F determines the computation partition of the DNN layers across mobile devices and the cloud [102].

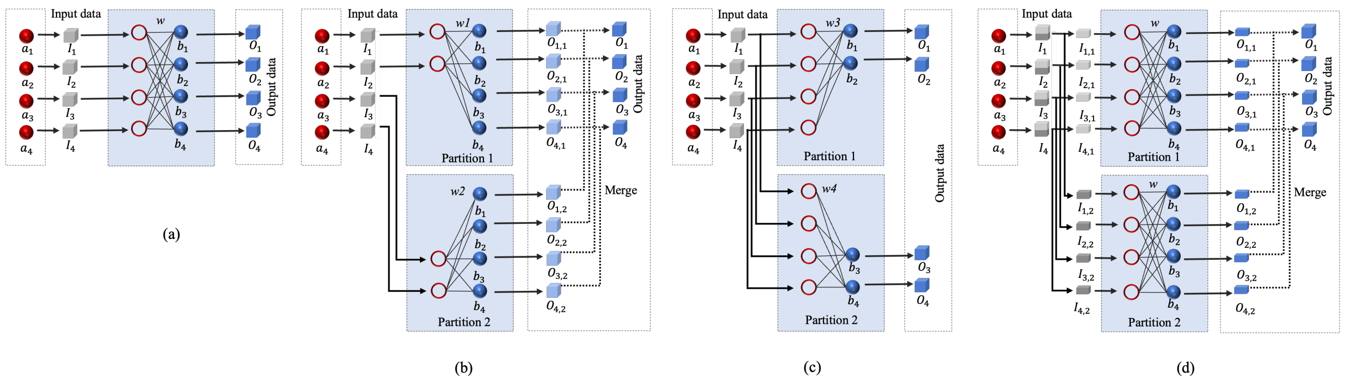
Fig. 11. Schematic diagram of neuron-level partition.

B. Neuron-level Partitioning

Neuron-level partitioning involves the segmentation of DLMs at the granularity of individual neurons such as weight matrices, filters, convolution kernels, etc. into several parts along a specific dimension. These partitions are then computed separately on end-edge-cloud nodes. There are two primary strategies for neuron-level partitioning: channel partitioning and spatial partitioning.

1) Channel partitioning: Channel partitioning strategies can be divided into input data channel partitioning and output data channel partitioning.

- Input data channel partitioning. As shown in Figure 11 (b), the output data $\{I_1, I_2, I_3, I_4\}$ produced by neurons $\{a_1, a_2, a_3, a_4\}$ in the upper layer is divided into several subsets, e.g. $\{I_1, I_2\}$, $\{I_3, I_4\}$, which are then transmitted to downstream computing nodes as input. Accordingly, each node possesses a crucial portion of the DLM weight parameters required to compute the output sub-dataset, e.g., $\{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,1}\}$ or $\{O_{1,2}, O_{2,2}, O_{3,2}, O_{4,2}\}$. Subsequently, the output sub-datasets from different nodes are aggregated (e.g., using a vector addition scheme), to form a complete output dataset $\{O_1, O_2, O_3, O_4\}$, which is then forwarded to the next layer. Taking a convolutional layer as an example, as shown in Figure 12, an RGB image serves as the input, consisting of 3 channels, corresponding to 3 kernels within each filter. The outputs of these 3 kernels are summed to generate the output feature map of the filter. If there are 4 filters, then 4 feature maps (4 channels) are produced as output. Input data channel partitioning for a convolution layer specifically involves partitioning the kernels. The input dataset is divided according to the number of computing nodes by channel, and each node uses the corresponding kernels to compute the input feature map. The feature maps are then aggregated for subsequent processing in the next layer.

- Output data channel partitioning. As shown in Figure 11 (c), this strategy involves transferring all the output data $\{I_1, I_2, I_3, I_4\}$ from neurons $\{a_1, a_2, a_3, a_4\}$ in the upper layer to computing nodes. Each node possesses an essential portion of weight parameters to generate a subset of output data corresponding to the input data, such as $\{O_1, O_2\}$ and $\{O_3, O_4\}$. The output data from all nodes are merged to obtain a complete output data $\{O_1, O_2, O_3, O_4\}$. In the context of a convolutional layer, output data channel partition essentially involves partitioning the filters. As shown in Figure 13, filters are distributed across two computing nodes, each of which possesses a full copy of the input data. These individual filters are used to process the input data and produce the corresponding feature maps. Subsequently, all feature maps are combined to form the input data for the next layer. The output data channel partitioning offers more advantages during DLM training as it reduces the communication cost associated with synchronizing network parameters. However, distributing input data to all computing nodes introduces significant additional communication overhead, which makes output data channel partitioning less practical for inference scenarios [180].

In practical applications, the channel partitioning strategy can be implemented in either a layer-wise or fused-layer manner. For example, DeeperThings [181] employs a two-layer

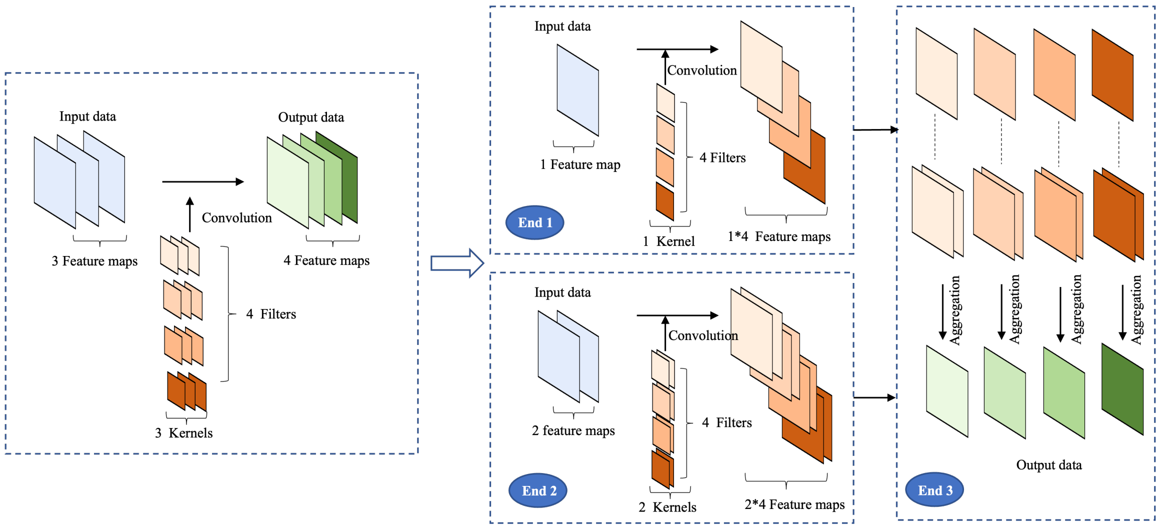

Fig. 12. Input data channel partitioning of a convolutional layer.

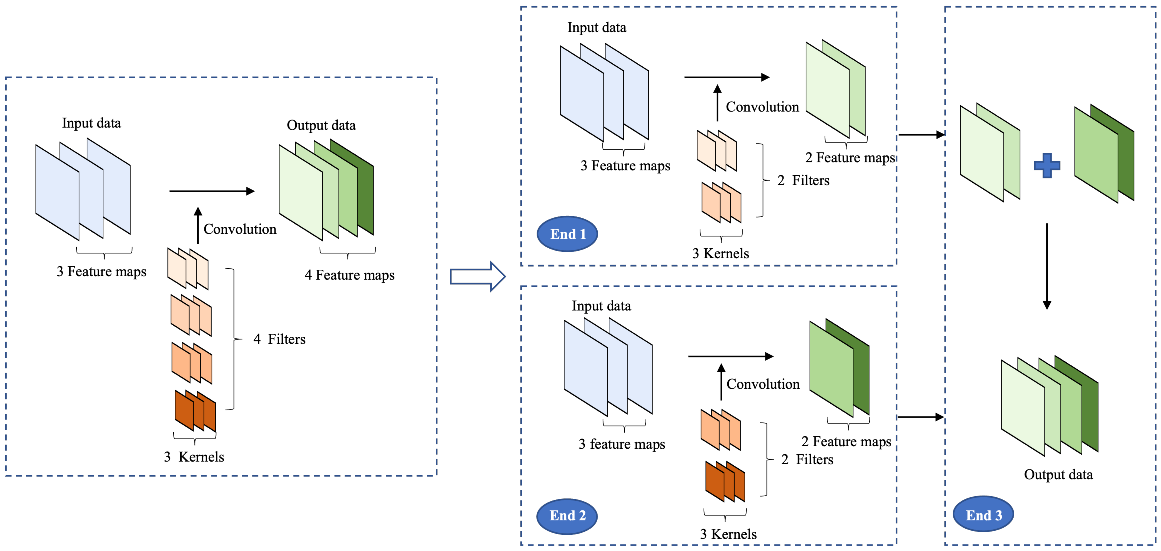

Fig. 13. Output data channel partitioning of a convolutional layer.

fusion approach, initially utilizing the output data channel strategy and subsequently employing the input data channel strategy for convolutional layers in a CNN during multi-device parallel computing. Compared to layer-wise channel partitioning, the fused-layer approach proves effective in reducing communication costs between devices since it only necessitates communication between nodes for the input of the first layer and the output of the last layer. However, it does impose higher demands on the computational and storage resources of the nodes.

2) Spatial partitioning: Spatial partitioning involves dividing the input or output data into several parts and processing these partitions across end-edge-cloud nodes. In this method, as shown in Figure 12 (d), the input data $\{I_1, I_2, I_3, I_4\}$ is split into segments, such as $\{I_{1,1}, I_{2,1}, I_{3,1}, I_{4,1}\}$ and $\{I_{1,2}, I_{2,2}, I_{3,2}, I_{4,2}\}$. Each of these segments is then transmitted to a computing node that possesses all the model parameters. These nodes process the partitioned input data to generate segmented output data, such as $\{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,1}\}$ and $\{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,1}\}$. Finally, the segmented output data from all nodes is combined to form the complete output data $\{O_1, O_2, O_3, O_4\}$. Spatial partitioning offers advantages in terms of reduced communication costs between devices, making it suitable for scenarios involving large input data. However, it requires each device to maintain a copy of the DLM, which may not be practical for resource-constrained devices with limited storage capacity.

This spatial partitioning method is commonly applied to convolutional layers, with two main partitioning rules: grid partitioning and vertical partitioning.

- **Grid partitioning.** As shown in Figure 14 (a), grid partitioning is used to divide the output data (feature maps) into n parts horizontally and vertically. By considering the size of the convolution kernel, as well as the dimensions and locations of the required input feature maps for each segment of the output feature map, the input data can be divided into ’n’

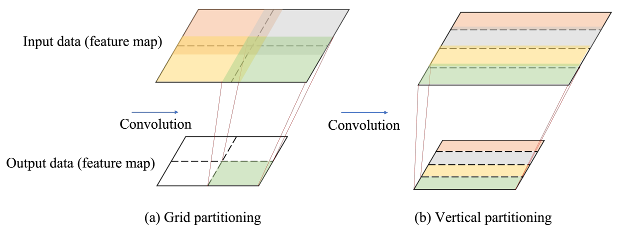

Fig. 14. Output data channel partitioning of a convolutional layer.

copies for distributed processing across convolutional layers.

- Vertical partitioning. As shown in Figure 14 (b), the vertical partitioning involves dividing the output data (feature maps) into n vertical segments. Vertical partitioning also relies on reverse inference based on the convolution kernel's size and the dimensions and positions of the required input feature maps for each output segment.

These partitioning strategies are commonly applied to convolutional layers to optimize distributed processing in DLMs. It is worth noting that, except for 1*1 convolution, both grid partitioning and vertical partitioning methods can result in overlapping portions of the input feature maps in Figure 14 (a) and (b). This overlap can lead to redundant communication costs. Specifically, for grid partitioning, if the output feature map (with a size of $m \times m$) is equally divided into n copies, the amount of overlapping data generated can be calculated using the formula $[sm + \sqrt{n}(f - s)]^2 - [sm + f - s]^2$. For vertical partitioning, the overlapping data can be calculated as $(sm + f - s) \cdot [sm + n(f - s)] - [sm + f - s]^2$. Here, the size of the convolution kernel is $f \times f$, and s is the step length, usually $f \geq s$. When n is the same, the overlap generated by grid partitioning tends to be smaller than that of vertical partitioning. Therefore, the choice between these spatial partitioning methods should be made carefully, taking into account the specific scenario to reduce the number of communication channels established between devices and decrease data transmission time.

Similar to channel partitioning, spatial partitioning strategies can be implemented as layer-wise or fused-layer methods. Multi-layer fusion approaches are advantageous as they can reduce communication costs between nodes and achieve significant computational acceleration. Mao et al. [109] designed the BODP (Biased One-Dimensional Partition) model partition method which segments each convolutional layer of VGG-16 using the vertical partitioning technique. Zhao et al. [110] designed DeepThings, a distributed CNN deployment framework for edge nodes, which utilized a grid partitioning method for fused layers to reduce memory usage and inter-device communication overhead for end devices. However, DeepThings did not optimize the fully connected layer in CNNs, which has been addressed in DeeperThings [181]. Zhou et al. [180] proposed an adaptive CNN acceleration framework that dynamically selects the optimal partition strategy based on the status of computational resources and network conditions. They used grid partitioning with convolutional layer fusion. Experimental results show speedups ranging from 1.9 to 3.7 times for three popular CNN models on 8 Raspberry Pi3 devices connected wirelessly.

C. Summary and Lessons Learned

1) End-to-end environment-aware layer-level partitioning: The choice of the cut points for a DLM using layer-level partitioning can vary depending on several factors, such as model size, available computing resources, the number of end-edge-cloud nodes, network conditions, and optimization objectives such as end-to-end inference latency, energy consumption, optimization time, etc. To determine optimal partition points, especially for DLMs with directed graph structures, graph theory analysis and path optimization methods can be applied. Special attention should be given to DLMs with directed cyclic graph structures, which require further exploration. Given the dynamic nature of power, storage, and network bandwidth in terminal devices, the development of lightweight and adaptive methods for end-to-end layer-level partitioning remains an ongoing challenge.

2) Customized neuron-level partitioning: Neuron-level partitioning is suited for larger DLMs with a larger number of layers, making it a useful approach for distributing one or more horizontal layers across resource-constrained end-edge-cloud nodes. This method can also be combined with layer-level partitioning for fine-grained tasks. However, due to the diversity of model structures and device types, a customized neuron-level partitioning approach is often required, possibly in conjunction with hardware optimization. Meanwhile, neuron-level partitioning may increase communication and data synchronization between computing nodes, making it more suitable for DLMs operating in robust network environments.

3) Future trend: The increasing complexity of DL model variants (e.g. dynamic DL which can conditionally skip or add the computation of network layers such as early exiting, layer skipping, and dynamic routing according to different tasks and inputs) and the endless emergence of edge-cloud intelligent collaboration paradigms pose challenges for efficient, dynamic and flexible partitioning and orchestrate large-scale DLMs in the end-edge-cloud framework. These issues can be solved with the architecture of deep learning-as-a-service (DLaaS) which is an emerging paradigm for developing service-oriented DL services and applications. There are several advantages of deploying DLaaS in end-edge-cloud CPN as follows.

- Modularity and personalization. DLaaS breaks down DL models into independent services or microservices, each focusing on a specific task or function. This modular nature makes DL models easier to manage and personalize to support dynamic DL and edge-cloud intelligent collaboration paradigms.

- Flexibility and scalability. Large-scale DL services can be customized and optimized to meet dynamic demands and resource availability. On the one hand, DLaaS enables the choosing of end-edge-cloud computing nodes that best suit each DL microservice. On the other hand, DL microservices can scale independently based on fluctuations in load.

- Independent and robustness. DL microservices can be deployed and updated independently, allowing upgrading or fixing individual models without affecting the entire system. Moreover, the robustness of DL systems can also be enhanced by the redundant deployment of DL microservices.

- Efficiency and fast response. DL microservices allow for reuse across multiple applications, saving development time and resources. Moreover, DLaaS enables the intelligence to sink to edges and terminals that are close to users for fast response and low latency capabilities.

Therefore, partitioning and orchestrating large-scale DLMs based on the DLaaS in the end-edge-cloud framework will become a major trend in the future.

VI. Knowledge transfer technologies for deep learning

Effective knowledge sharing plays a pivotal role in facilitating mutual learning and collaborative updates across end-edge-cloud systems. This section focuses on knowledge transfer techniques within the realm of DL, including transfer learning and knowledge distillation.

A. Transfer Learning

Transfer learning is the practice of leveraging existing knowledge from a source domain or task to enhance learning in a target domain or task. The effectiveness of transfer learning is often contingent on the degree of correlation between the source and target domains or tasks. Identifying common knowledge shared between these domains is paramount to successful transfer learning. Depending on the nature of the knowledge, as shown in section III-C, end-edge-cloud transfer learning can be classified into the following types.

1) Data-based transfer learning.: It aims at sharing source domain knowledge including instance knowledge, data representation knowledge, and data relational knowledge.

- Instance knowledge-based transfer learning. It carefully selects relevant samples from the source domain dataset to assist training within the target domain. Weighting and importance sampling are two major strategies. For example, the ITrAdaBoost algorithm [182] is proposed to adjust the weights of a wrongly classified instance in a source domain according to the distribution distance from the instance to a target domain, and these weighted samples are then integrated with samples from the target domain to train the target domain DLM.

- Data representation knowledge-based transfer learning. It aims to find the common feature space of the source domain and target domain, including transforming the source features to match the target ones and transforming both the source and the target features into a new feature representation. By doing so, it facilitates the utilization of existing labeled data samples from the source domain for training within this new feature space, enhancing transfer learning capabilities.

- Data relational knowledge-based transfer learning. It uses the correlations between instances, often in the form of logical relationships or rules, derived from both the source and target domains. Such correlations are harnessed to facilitate knowledge transfer between domains, thereby enhancing the learning process in the target domain.

2) Model knowledge-based transfer learning.: This form of transfer learning involves the utilization of pre-existing knowledge from a source domain DLM, which encompasses parameters or prior hyperparameters, categorized as structured knowledge (as described in Section III-C). In essence, a model that has been pre-trained in the source domain for one task can be fine-tuned or adapted to perform another specific task in the target domain. For example, structured knowledge such as weight parameters from a cloud-DLM can be shared with edge- or end-based DLMs that are further trained or personalized using new local data. In addition, model knowledge extends beyond structural knowledge and encompasses feature knowledge and model relational knowledge. However, the transfer of these types of model knowledge typically falls outside the scope of classical transfer learning and is often achieved through the application of knowledge distillation techniques.

3) Advanced transfer learning technologies.: From a technical perspective, numerous innovative techniques and methods have emerged. While the space limitations prevent an exhaustive discussion of these techniques, some of the most recent advancements in transfer learning can be found in the literature [44], [183], [184].

- Meta-learning. It aims to enhance generalization performance across multiple tasks or domains by effectively learning "how to learn". This involves the automatic selection of appropriate model architectures, optimization algorithms, and hyperparameters, facilitating rapid achievement of strong performance on new tasks.

- Transfer reinforcement learning. Combining techniques from transfer learning and reinforcement learning, this approach leverages shared knowledge and experience from reinforcement learning across diverse tasks. This leads to accelerated learning and optimization of policies for the target task.

- Transfer generative adversarial networks. Drawing inspiration from generative adversarial networks, this method employs transfer learning to map data from the source domain to the target domain. This mapping enables data augmentation and sample generation in the target domain.

- Incremental transfer learning. This approach aims to achieve progressive transfer learning by introducing new target tasks continuously. The DLM accumulates knowledge through gradual learning and adapts to an increasing number of target tasks over time.

- Multimodal transfer learning. Designed to handle multimodal data, this technique facilitates the transfer and sharing of information across different data modalities. It enables comprehensive learning for the target task by leveraging insights from diverse sources.

Overall, the transfer of source domain data typically incurs higher privacy and communication costs compared to the transfer of model structure and parameters. Therefore, model knowledge-based transfer learning methods are often more suitable for the context of end-edge-cloud co-updating. Meanwhile, these advanced transfer learning technologies each have unique advantages and applications, providing robust technical support for diverse and personalized end-edge-cloud co-updating scenarios.

B. Knowledge Distillation

Knowledge distillation, initially proposed for model compression [185], serves the purpose of transferring knowledge from a large "teacher" model to a compact "student" model. Recently, it has found applications in model enhancement, focusing on improving the performance of a DLM by distilling other DLMs' knowledge, such as mutual distillation or self-distillation. Unlike transfer learning, which emphasizes the knowledge from domain data, knowledge distillation primarily emphasizes model knowledge including feature knowledge (intermediate or output feature knowledge), model structure knowledge, and model relational knowledge, as shown in Figure 6. For example, the concept of classical knowledge distillation involves training the student model to replicate the outputs of the teacher model when presented with the same input. This process aims to enable the student model to learn the representation capability inherent in the teacher model.

1) Knowledge distillation in end-edge-cloud co-learning: The configuration of the knowledge distillation process, involving a "teacher" and a "student" model, aligns exceptionally well with the end-edge-cloud architecture. The "teacher" model and "student" model can be deployed across end-edge-cloud computing nodes to achieve co-updating. For example, the smaller "student" model can be deployed on edge or end nodes, while the larger "teacher" model resides in the cloud, guiding the training of the "student" model. Recent studies have established the feasibility of knowledge distillation in end-edge-cloud co-learning. Knowledge distillation technologies applicable in the end-edge-cloud context include multi-teacher learning, knowledge amalgamation, teacher assistant, cross-modal distillation, and mutual distillation.

- Multi-teacher learning. It involves using the knowledge acquired by multiple teacher models in the context of a single student model. A simple approach is to utilize soft labels or intermediate/output feature maps generated by multiple "teacher" models through voting, random, or weighting policies [186], [187]. This contributes to the enhancement of the "student" model's performance. Meanwhile, the complementary knowledge [188], [189] possessed by "teacher" models can be used to guide the training of the "student" model.

- Knowledge amalgamation. It is to transfer knowledge from multiple "teacher" models (multiple tasks), often associated with different tasks, to a single "student" model. For example, soft-label knowledge with the highest confidence from multiple "teacher" models can be used to train and update student models [190]. Alternatively, feature maps from multiple "teacher" models can be fused to facilitate the training and updating of the student model [191], [192].

- Teacher assistant. It refers to the utilization of intermediate models to assist the student model in acquiring knowledge from the teacher model [193]. This approach effectively bridges the gap between the "teacher" and "student" models regarding the large difference in model capacity. Within the end-edge-cloud framework, a medium-sized model (smaller than the "teacher" model but larger than the "student" model) can be deployed on the edge to serve as a teacher assistant. It first learns from the "teacher" model in the cloud and then transfers the acquired knowledge to the "student" model on the end device.

- Cross-modal distillation. It involves training the teacher model using labeled data from different modalities and then employing the teacher model to instruct the student model using information from other modalities. This facilitates knowledge distillation across modalities. For example, in emotion recognition, the soft labels generated by an image-based emotion recognition model ("teacher") can guide the training of a speech-based emotion recognition model ("student") [194]. Cross-modal distillation effectively utilizes multimodal sample data but faces challenges related to modal alignment.

- Mutual distillation. It is to enable a group of untrained "student" models to learn from each other [195]–[197]. The knowledge acquired includes feature knowledge (such as intermediate or output features), model structured knowledge, and model relational knowledge. Mutual distillation holds significance as it allows "student" models to mutually learn from each other to enhance the model performance and speed up the training process [197].

Both knowledge amalgamation and multi-teacher learning belong to the category of the "multi-teacher to single-student" training mode. Their difference lies in their objectives: knowledge amalgamation uses multiple "teacher" models to equip the "student" model with the capability to handle multiple tasks, thereby enhancing its overall generalization ability. In contrast, multi-teacher learning aims to improve the performance of the "student" model on a single specific task.

In general, knowledge distillation techniques offer extensive methodological and technical support for end-edge-cloud co-updating. Recent research advances in knowledge distillation can be found in the literature [46], [47], [198].

2) Federated knowledge distillation: Knowledge distillation has been leveraged as an important technique to address the challenges associated with communication and computing efficiency, heterogeneity, and personalization in FL.

- Resource-efficient federated knowledge distillation. On the one hand, in the classic FL paradigm, the parameter server and workers need to intensively communicate the model update information, resulting in huge communication costs, especially when the model contains numerous parameters. Knowledge distillation has been introduced in FL to tackle this issue. For example, in FedKD [199], there is a small model that learns from a large global model through mutual knowledge distillation, which is shared by different clients instead of directly communicating large models between the clients and the parameter server. Itahara et al. [200] proposed a distillation-based semi-supervised FL algorithm that exchanges the outputs of local models on the public dataset, instead of model parameters among mobile devices to achieve efficient communication. Similarly, Sattler et al. [201] introduced quantization and delta coding to the exchanged outputs (soft labels) of the local models. On the other hand, a series of studies [202]–[204] focused on training large models on resource-constrained devices. For example, He et al. [202]

designed a variant of the alternating minimization approach to train small CNNs on edge nodes and periodically transfer their knowledge through knowledge distillation to a large server-side CNN.

- Heterogeneous federated knowledge distillation. Most of the current FL algorithms require homogeneous local devices with the same on-device models. Given the widespread yet heterogeneous devices in the end-edge-cloud environment, studies [205], [206] aim to leverage knowledge distillation to enable FL training heterogeneous on-device models to adapt to the heterogeneous devices. Yu et al. [205] proposed a resource-aware FL method that aggregated local knowledge from edge models that are suitable for heterogeneous devices with different computing power. Zhang et al. [206] introduced a federated learning framework with heterogeneous on-device models through zero-shot distillation.

Other studies aim to tackle the data heterogeneity and personalization issue in FL. On the one hand, studies [207]–[212] devoted to addressing the Non-IID challenge for the global model, by distilling the knowledge such as soft targets from multiple teacher models trained by different clients to the global server model. Lin et al. [207] leveraged the unlabeled public dataset to obtain the soft labels (logits) from multiple local client models and then updated the global model with these soft labels. Similarly, Zhu et al. [208] and Zhang et al. [209] improved this federated knowledge distillation approach by replacing the public shared dataset with data generated by the generative model. However, these soft labels can be wrong, resulting in a performance degradation of the global model. Thus, DaFKD [210] was proposed to discern the importance of each client model's output to reduce the impact of wrong soft labels. In addition, FedGKD [211] fused the knowledge from historical global models and guided local training to alleviate the client-drift issue that could considerably impede the convergence of global model training. On the other hand, studies [213], [214] aim to realize the client's model personalization. In FedICT [213], a bi-directional distillation framework is proposed to reinforce the clients' fitting of local data while alleviating client drift issues. Jin et al. [214] proposed a personalized federated learning framework via self-knowledge distillation, which allows clients to distill the knowledge of previous personalized models to current local models.

Most of the above approaches require that the data distribution of publicly available datasets should closely resemble that of local training data at clients. However, it is not easy to get or generate such datasets whether labeled or not. Meanwhile, it is important to note that the favorable properties of all federated knowledge distillation methods come at the cost of additional computational overhead caused by knowledge distillation. This additional computational overhead might be challenging to resource-limited client devices. In addition, these approaches also exchange model parameters or knowledge such as logits recursively, resulting in privacy issues. Gong et al. [215] developed a privacy-preserving and communication-efficient method in an FL framework with quantising and adding noise on logits for aggregation and distillation.

C. Efficient Fine-tuning of Foundation Models

Foundation models are trained on massive, diverse, and unlabeled datasets, typically through self-supervised learning, and can be applied to numerous downstream tasks [216], [217]. On a technical level, foundation models are enabled by transfer learning where a model is trained on a source domain and then adapted to the downstream task (target domain) via fine-tuning [216]. This subsection introduces two branches of representative efficient fine-tuning approaches, including parameter-efficient fine-tuning and resource-efficient fine-tuning.

1) Parameter-efficient fine-tuning: Full parameter fine-tuning that updates all the parameters with different instances for different tasks, is rendered impractical for large-scale foundation models due to the high cost of computation and storage [218]. This necessitates a branch of research on parameter-efficient fine-tuning that aims to adapt a foundation model to downstream tasks by updating a small portion of the model parameters to reduce computation and storage overhead. One simple approach involves manually or heuristically adapting a small set of parameters of the foundation models to different downstream tasks. For example, BitFit [219] froze most of the transformer-encoder parameters and trained only the bias-terms and the task-specific classification layer, which could still reproduce over 95% performance on several benchmarks. However, this approach relies on expert experience and requires targeted design for specific foundation models. Recent studies have introduced a modest amount of trainable parameters, facilitating domain adaptation for pre-trained extensive models by optimizing these additional parameters on tailored datasets, significantly diminishing computational overhead and lowering the barriers to fine-tuning foundational models. These approaches mainly include adapter-based tuning, prompt-based tuning, and low-rank adaptation.

- Adapter-based tuning. Adapters are small-scale and trainable neural modules that can be integrated into foundation models' layers. For example, the adapters are separately inserted after the multi-head self-attention and the feed-forward network sublayers in the transformer layer, and only these adapters are tuned for domain adaptation. A simple instantiation of the adapter contains a down-projection and an up-projection [220]. The down-projection converts the input d-dimensional feature $h \in \mathbb{R}^d$ to a r-dimensional space with a parameter matrix $W_d \in \mathbb{R}^{d\times r}$ and is followed by a non-linear function $f(\cdot)$. Then the up-projection W_u maps the r-dimensional representation back to d-dimensional space and is added with a residual connection. As the adapter has demonstrated impressive parameter tuning efficiency and robustness [221], and thus has been widely adopted. Efforts [222]–[225] have been devoted to designing efficient adapters and placement strategies. For example, AdaMix [223] can leverage a mixture of adapters like Houlsby or a mixture of low-rank decomposition matrices like LoRA to improve downstream task performance. AdapterDrop [224] removes adapters from lower transformer layers during training and inference.

- Prompt-based tuning. It injects trainable prompt tokens with additional context to the original input. At present, these trainable prompt tokens can be incorporated into the input layer (known as prompt-tuning [226]) and each of the intermediate layers (known as prefix-tuning [227]). Prefix-tuning [227] adds trainable continuous prefix tokens to each layer in a large language model, and keeps language model parameters frozen while optimizing these continuous prefix tokens. Similarly, LLaMA-Adapter [228] adopts a set of learnable adaption prompts as prefixes to the input word embeddings at higher transformer layers. Compared with prefix-tuning, prompt-tuning [226] is a more simplified strategy that only injects soft prompts to the input layer such as soft prompt [226], P-tuning [229], and P-tuning V2 [230]. Recent advancements explore how soft prompts could be used for few-shot [231], pre-training [232], or multitask [233].

- Low-rank adaptation. It leverages low-rank representations to minimize the number of trainable parameters. For example, LoRA [234] freezes the pre-trained model weights and injects two trainable rank decomposition matrices $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$ into each layer of the Transformer architecture to tuning the weight matrix $W \in \mathbb{R}^{m\times n}$ as $W \leftarrow W + A \cdot B$. This approach is straightforward to implement and decreases the trainable parameter count while still allowing the tuning of high-dimensional matrices.

Overall, the parameter-efficient fine-tuning is computational efficiency, for example, the training of adapters could be 60% faster than vanilla fine-tuning [218]. However, both the adapter-based tuning and the prompt-based tuning have the problem of increasing inference delays. The adapter-based tuning adapts the pre-trained model to downstream tasks through the adapter module, which adds additional model parameters and brings about inference delay issues. For prompt-based tuning, increasing the length of the input by 20-100 tokens can significantly increase computation [235]. In addition, prompt-based tuning methods have the problem of difficulty in optimization, as they converge slower than full parameter fine-tuning and other parameter-efficient fine-tuning methods [218]. Comparatively, the low-rank adaptation stands out as it does not introduce additional inference latency, exhibits a reduced training threshold, and demonstrates robust compatibility. This approach is congruent with the majority of efficient parameter fine-tuning algorithms, allowing for its integration to further enhance the performance of large-scale models on novel tasks.

2) Resource-efficient fine-tuning: As foundation models scale up, the computation and memory needed for fine-tuning is dramatically increased, which limits the fine-tuning efficiency for resource-constrained end-edge-cloud devices. Consequently, minimizing computation and memory usage in fine-tuning has emerged as a critical topic. One approach is based on quantization [236]–[239]. For example, QLoRA [236] quantizes the model parameters into 4-bit and fine-tunes this quantized model with a low-rank adapter, which reduces memory usage to fine-tune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. Another approach is based on gradient [240], [241]. For example, Malladi et al. [241] adapted the classical zeroth-order-SGD method to operate in-place, thereby fine-tuning language models with the same memory footprint as inference. In addition, LoRA-FA [242] is a memory-efficient fine-tuning method that chooses to freeze the projection-down weight and update the projection-up weight in each LoRA layer.

D. Summary and Lessons Learned

1) Knowledge distillation versus transfer learning: Both knowledge distillation and transfer learning involve the transfer of knowledge. However, knowledge distillation and transfer learning respectively emphasize the transfer of knowledge from models and the transfer of knowledge from data. The distinctions and connections between knowledge distillation and transfer learning are outlined as follows.

(i) The concept of transfer learning is relatively broad, including both data-based transfer and model-based transfer, while knowledge distillation is essentially a specific form of model-based transfer learning. In essence, knowledge distillation serves as a means to achieve transfer learning.

(ii) Knowledge distillation primarily serves the purpose of compressing or enhancing the "student" DLM, focusing on model size and performance improvements. In contrast, transfer learning typically does not address the lightweight model issue and is more oriented towards addressing challenges like limited labeled data or domain adaptation using data or model knowledge from other domains.

(iii) In transfer learning, the target and source domains can be homogeneous or heterogeneous, and learning often takes place on different datasets, addressing tasks like domain adaptation. However, knowledge distillation typically operates within the same dataset, with a focus on enhancing or compressing the model.

(iv) Knowledge distillation mainly involves model knowledge which can originate from one or multiple homogeneous or heterogeneous DLMs, typically focusing on the feature knowledge of the model. However, conventional transfer learning methods primarily use knowledge from other domains' data to train models in the target domain and rarely use model knowledge.

2) Efficient fine-tuning: Parameter-efficient fine-tuning has demonstrated its practicality. However, such kind of methods are still not well understood and are highly sensitive to hyperparameters. Moreover, existing methods introduce trainable parameters to different tasks depending solely on human heuristics and neglect the domain gaps [243]. Therefore, matching the performance of full fine-tuning approaches remains a challenge. In addition, edge and end nodes have constraints in memory, computation, and even energy. To mitigate these challenges, resource optimization for fine-tuning can be effectively achieved through the integration of techniques such as quantization and pruning.

3) Future trend: DL systems operating in the real world are exposed to continuous streams of information and thus are required to continuously learn and remember multiple tasks from dynamic data distributions. Continual learning, which refers to the capability of a DLM to continuously acquire new knowledge and adapt to new data without forgetting previously learned knowledge, has become the mainstream trend in the

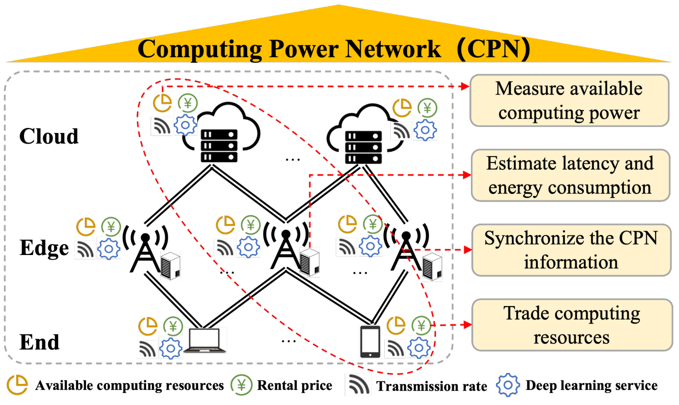
Computing Power Network (CPN)
Cloud
Edge
End
Measure available computing power
Estimate latency and energy consumption
Synchronize the CPN information
Trade computing resources
Available computing resources
Rental price
Transmission rate
Deep learning service

Fig. 15. Challenges of end-edge-cloud computing power network for deep learning.

future. However, traditional knowledge transfer technologies don't care about catastrophic forgetting and continuous learning. Therefore, continuous learning with knowledge transfer will become the future trend.

VII. Challenges and Prospects of End-Edge-Cloud Collaborative Deep Learning

A. Challenges of End-Edge-Cloud Collaborative Deep Learning Systems

The end-edge-cloud collaborative DL system faces several challenges, including diverse application requirements, unbalanced data distribution, heterogeneous computing resources, a dynamic network environment, and complex communication protocols. These challenges impact the optimization, adaptability, and reliability of the system.

1) Optimization: A significant challenge is efficiently achieving unified scheduling and optimization of distributed computing power, network, data, and applications within the end-edge-cloud system. This must strike a balance between latency, accuracy, and energy efficiency while considering different application requirements.

CPN [244] is a novel paradigm that further evolves toward the cooperation and integration of computing power and networks. It aims to provide ubiquitous end-edge-cloud computing, storage, and network resources to offer everywhere, intelligent, low latency, and flexible services for various applications [245]. However, the CPN is still in its early stage, and several open issues need to be addressed before it can realize its full potential as shown in Figure 15, including (i) How to measure computing power? (ii) How to estimate the computational latency and energy consumption of an application on a specific resource platform? (iii) How to quickly synchronize the CPN information? and (iv) How to efficiently trade computing resources?

2) Adaptability: The adaptability of end-edge-cloud DLMs is crucial to cater to fast-changing scenarios and diverse application requirements. Potential challenges include overfitting, imbalanced data, domain adaptive, and few-shot learning. Addressing these challenges requires the design of efficient model update methods such as transfer learning and federated knowledge distillation. In addition, dynamic neural networks [246] offer promise as a solution. These networks can adjust their structures or parameters based on varying inputs, making them more efficient, adaptable, compatible, generalizable, and interpretable compared to static models [246]. For example, Zhong et al. [247] proposed a DNN model capable of dynamically adjusting to changes in data distribution by altering its structure. Future research can focus on architecture design, optimal methods, and generalization for dynamic neural networks.

In addition, the adaptability of the system involves managing the heterogeneous end-edge-cloud nodes. Challenges in this regard include how to design customized models for different end-edge-cloud nodes, and how to facilitate the rapid development and deployment of large-scale DL applications. There are some potential solutions: (i) Integrating various technologies such as model compression, model segmentation, and dynamic neural networks to tailor suitable models for specific hardware platforms. (ii) Integrating microservices and virtualization technologies to achieve rapid distribution and deployment of DLMs.

3) Reliability: Due to the complex network environment and the diversity of end devices, how to ensure the stability and reliability of the end-edge-cloud system under cross-network and cross-region environments is very important, especially for many industrial applications that require ultra-low latency and high reliability. Therefore, exploring intelligent monitoring and maintenance technologies for cloud-edge-end systems is essential. The utilization of real-time monitoring mechanisms across end-edge-cloud infrastructures, coupled with the application of advanced AI techniques for the analytical processing of amassed monitoring data, facilitates the prognostication of incipient system failures, thereby preemptively addressing potential disruptions and enhancing the overall system resilience and reliability.

B. Communication Bottleneck among Heterogeneous End-Edge-Cloud Nodes

The successful implementation of end-edge-cloud co-learning depends on efficient interconnection between geographically distributed end-edge-cloud computing nodes through high-speed networks. Although current interconnection networks such as Gigabit/10 Gigabit Ethernet and Infiniband networks can offer impressive speeds of up to 100 Gb/s, there remains a noticeable disparity compared to the rapid data exchange rates achievable within the internal memory. Moreover, as the number of end-edge-cloud nodes grows, the associated communication overhead increases, potentially leading to a "bucket effect", wherein the overall system performance is limited by the node with the slowest network bandwidth. Therefore, when designing a heterogeneous cluster for end-edge-cloud co-learning, it is necessary to consider the impact of network bandwidth, latency, and jitter on system performance.

There are some potential solutions: (i) Data compression. A common method is to compress the data transmitted between nodes, including lossless compression and lossy compression. Lossless compression, although offering a limited compression ratio, typically does not impact the performance of DL algorithms. Lossy compression can achieve higher compression ratios but may affect the usability and effectiveness of the compressed data. Therefore, how to strike a balance between reducing communication costs and preserving data utility becomes a key issue. (ii) Sixth-generation mobile communication technology (6G): Transitioning to 6G can significantly improve the communication capabilities of end-edge-cloud co-learning systems. Unlike previous generations, 6G aims to provide not only enhanced communication performance but also greater coverage, increased computing power, and faster sensing efficiency. It integrates communication, sensing, and computing to deliver real-time and ultra-reliable communication services for end-edge-cloud DL applications. The development of an integrated space-air-ground network can ensure seamless and ubiquitous network access.

C. Self-adaptive Compression and Acceleration of Deep Learning Models

The adaptivity of end-edge-cloud DLMs signifies their ability to dynamically adjust and optimize compression techniques based on specific application demands, the available hardware resources, and the prevailing network environment on the target hardware platform. The following challenges still need to be addressed before self-adaptation can be achieved.

(i) How to evaluate the quality of compression methods is a key challenge. Existing evaluation criteria often compare performance indicators between post-compression and pre-compression models, such as TOP-1, TOP-5, acceleration ratio, storage space saving rate, compression ratio, etc. While Cheng et al. [248] have given some suggestions to select compression methods for a particular task, differences in hardware platforms used for experiments can complicate the unified evaluation of various compression methods. Moreover, in the dynamic end-edge-cloud framework, where tasks, available computing resources, and network environments change frequently, automatically selecting efficient compression methods and compression ratios becomes challenging. Therefore, there is a need for the development of more accurate evaluation models that can guide the selection of compression methods, accounting for varying environmental factors. (ii) While many compression methods are designed primarily for CNN, it's important to consider the adaptability of these methods to other types of DLMs, such as RNNs, GNNs, and Transformer-based models. It remains to be explored whether traditional compression methods are adaptable to these networks.

In addition to model compression, advanced model architectures and acceleration techniques such as quantum neural networks, optical neural networks, and superconducting neural networks should be investigated further. For example, optical neural networks [249], [250] harness photonic hardware acceleration for complex matrix-vector multiplications, offering advantages such as high bandwidth, rapid calculation speeds, and extensive parallelism compared to traditional electronic counterparts.

D. Self-adaptive Partitioning of a Deep Learning Model

In the end-edge-cloud framework, it's crucial to determine the optimal partition strategy and partition points for a DLM based on the available computational resources, network conditions, and specific tasks. Moreover, the number of end and edge nodes also changes dynamically. Selecting the most suitable model partition strategy for different end-edge-cloud environments poses a significant challenge. Therefore, to make full use of these devices, it's essential to dynamically adjust the number of nodes, the partition strategy, and partition points to ensure the scalability and adaptability of end-edge-cloud co-learning, especially in the context of the rapid growth of IoT and mobile devices.

In addition, many existing studies assume that DLMs are static and cannot adapt to changing inputs or device capabilities. However, as dynamic networks become more prevalent, the static network segmentation approaches may become less effective. Hence, future research should focus on developing self-adaptive partitioning technologies for dynamic networks.

E. Effective Knowledge Transfer in End-Edge-Cloud Collaborative Deep Learning

In knowledge-based end-edge-cloud co-updating methods, knowledge can include instance knowledge such as relations between sample data, and model knowledge such as DLM parameters, intermediate features, and relations between layers. However, several challenges need to be addressed in this context: (i) There is a lack of unified metrics and a comprehensive knowledge framework in this context. Existing knowledge transfer techniques are predominantly based on experimentation or empirical approaches. (ii) Effective theoretical explanations are needed to better understand the selection of knowledge to migrate and the choice of transfer techniques for optimal outcomes. (iii) Negative transfer occurs when transferred knowledge has a detrimental impact on the target learners [183]. Strategies to mitigate these negative effects and achieve successful knowledge transfer learning require further investigation.

F. Security and Privacy of End-Edge-Cloud Collaborative Deep Learning

The end-edge-cloud architecture utilizes diverse computing resources to fulfill the real-time, accurate, and resilient requirements of DL tasks. However, this distributed and collaborative computing paradigm also gives rise to significant security and privacy concerns. On the one hand, within the end-edge-cloud collaborative framework, data regularly traverses between different layers and various devices. Some of these end-edge-cloud nodes possess weak security capabilities, rendering them susceptible to potential attacks. These security vulnerabilities could compromise data integrity, availability, and overall system security. On the other hand, the end-edge-cloud collaborative system often involves multiple organizations with varying degrees of trust between them. Such malicious actors could undermine the transparency and security of the collaborative computing process, potentially leading to various security breaches.

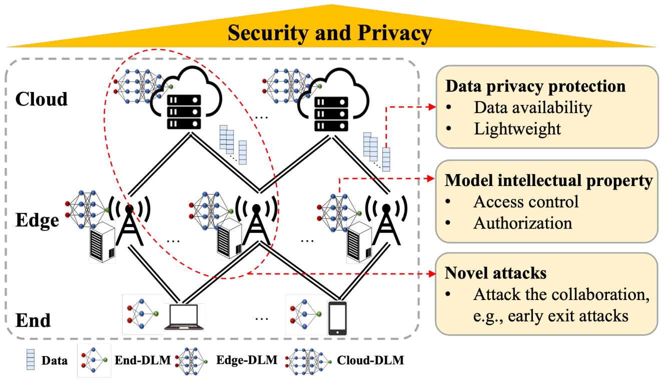

Fig. 16. Security and privacy challenges of end-edge-cloud collaborative deep learning.

Therefore, as shown in Figure 16, (i) in the pursuit of effective end-edge-cloud solutions, it becomes imperative to incorporate privacy protection technologies, including differential privacy, secure multi-party computing, encryption, and blockchain [251]. However, data availability and privacy protection are a pair of contradictions. In particular, DLMs have complex structures and lack interpretability, rendering the delicate balance between privacy protection and data availability challenging to quantify. Therefore, a key research issue revolves around achieving a tradeoff between data availability and privacy protection to meet application demands. Moreover, the substantial computational overhead associated with current privacy protection methods presents a formidable obstacle when deploying models on resource-constrained end devices. Consequently, it becomes imperative to develop a more streamlined privacy or encryption algorithm within the framework of the end-edge-cloud co-learning architecture. (ii) The proficiently trained model can be disseminated across the end-edge-cloud nodes, potentially leading to unrestricted access and the risk of infringement upon the model owner's intellectual property rights. Consequently, the exploration of model authorization emerges as a crucial area for future research. For example, methods like backdoor watermarking can be employed to enhance the protection of the model's intellectual property. (iii) End-edge-cloud co-learning introduces a novel perspective on model attack. This novel attack paradigm doesn't primarily aim to undermine the DLM's accuracy. Instead, its focal point is to disrupt the collaborative mechanism or reduce the efficiency of end-edge-cloud collaboration. For example, early exit attacks [252] have the potential to hinder the progressive inference mechanism. Hence, both the attack strategies and safeguards for the end-edge-cloud collaborative mechanism emerge as an inevitable research trajectory.

VIII. Conclusion

Time-sensitive personalized DL requirements and the rapid development of end and edge computing push researchers and enterprises to embrace the end-edge-cloud collaborative computing paradigm. This paradigm plays an important role in enhancing the DL performance but is still in its nascent stages of success. To comprehensively present the foundational principles of end-edge-cloud co-learning, we systematically review the progress in this direction and attempt to build a comprehensive research framework. Specifically, in this paper, we conduct a systematic analysis of collaboration elements within end-edge-cloud DL from both a system and collaboration perspective. These elements include data, model, and computing power considerations. We also identify potential manners and mechanisms for end-edge-cloud co-learning, such as distributed and collaborative training, inference, and model updating. Then, we highlight the details and progress of enabling technologies of end-edge-cloud co-learning. These technologies include model compression and lightweight technologies tailored for end and edge computing, model partition methodologies for end-edge-cloud co-inference, and knowledge transfer strategies for end-edge-cloud co-updating. Finally, we discuss problems and challenges that still exist in end-edge-cloud co-learning and the possible research directions in the future.

We anticipate that with the rapid evolution of end-edge-cloud computing platforms and the continuous expansion of DL applications, end-edge-cloud collaboration will mature. This evolution will enable the resolution of real-time and personalized DL challenges, thereby generating significant value for various industries. We hope this survey will stimulate further discussions and research efforts aimed at advancing the integration of DL and end-edge-cloud computing within this promising paradigm.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85-117, 2015.

[3] Y. Bengio, Y. Lecun, and G. Hinton, "Deep learning for AI," *Commun. ACM*, vol. 64, no. 7, pp. 58-65, 2021.

[4] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Comput. Sci. Rev.*, vol. 40, p. 100379, 2021.

[5] F. Gu, M. H. Chung, M. Chignell, S. Valaee, B. Zhou, and X. Liu, "A survey on deep learning for human activity recognition," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1-34, 2022.

[6] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4316-4336, 2020.

[7] A. C. De Araujo and A. Etemad, "End-to-end prediction of parcel delivery time with deep learning for smart-city applications," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 17043-17056, 2021.

[8] C. Yang, Y. Wang, S. Lan, L. Wang, W. Shen, and G. Q. Huang, "Cloud-edge-device collaboration mechanisms of deep learning models for smart robots in mass personalization," *Robot. Comput. Integr. Manuf.*, vol. 77, p. 102351, 2022.

[9] T. B. Brown et al., "Language models are few-shot learners," in *Proc. 34th Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2020, pp. 1877-1901.

[10] S. Wu et al., "Yuan 1.0: Large-scale pre-trained language model in zero-shot and few-shot learning," 2021, [Online]. Available: http://arxiv.org/abs/2110.04725

[11] A. Chowdhery et al., "Palm: Scaling language modeling with pathways," *J. Mach. Learn. Res.* vol. 24, no. 240, pp. 1-113, 2023.

[12] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: a general framework of collaborative edge-cloud AI," in *Proc. 27th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, Aug. 2021, pp. 2543-2553.

[13] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Comput.*, vol. 8, no. 4, pp. 14-23, 2009.

[14] V. Va, T. Shimizu, G. Bansal, and R. W. Heath, "Millimeter wave vehicular communications: A survey," *Found. Trends Netw.*, vol. 10, no. 1, pp. 1-116, 2016.

[15] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96-101, 2018.

[16] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4950-4960, 2020.

[17] E. Tartaglione, A. Fiandrotti, S. Lepsy, and G. Francini, "Learning sparse neural networks via sensitivity-driven regularization," in *Proc. 32th Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018, pp. 3878-3888.

[18] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4013-4021.

[19] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: http://arxiv.org/abs/1704.0486.

[20] K. S. Zaman, M. B. I. Reaz, S. H. M. Ali, A. A. A. Bakar, and M. E. H. Chowdhury, "Custom hardware architectures for deep learning on portable devices: A review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 33, no. 11, pp. 6068-6088, 2022.

[21] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1-35, 2023.

[22] F. Saeik et al., "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Comput. Networks.* vol. 195, p.108177, 2021.

[23] P. McEnroe, S. Wang, and M. Liyanage, "A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15435-1545935, 2022.

[24] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali, "Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey," *IEEE Commun. Surv. and Tutorials*, vol. 25, no. 2, pp. 1199-1226, 2023.

[25] T. Le Duc, R. G. Leiva, P. Casari, and P. O. stberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1-39, 2019.

[26] S. Ahmad, I. Shakeel, S. Mehfuz, and J. Ahmad, "Deep learning models for cloud, edge, fog, and IoT computing paradigms: Survey, recent advances, and future directions," *Comput. Sci. Rev.*, vol. 49, p. 100568, 2023.

[27] K. Zhang et al., "Compacting deep neural networks for Internet of things: Methods and applications," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 11935-11959, 2021.

[28] T. Zhao et al., "A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities," *Proc. IEEE*, vol. 110, no. 3, pp. 334-354, 2022.

[29] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable tinyml," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1-30, 2023.

[30] G. Menghani, "Efficient Deep Learning: A Survey on making deep learning models smaller, faster, and better," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1-37, 2023.

[31] M. G. Sarwar Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 137, 2022.

[32] J. Zhang et al., "Edge learning: The enabling technology for distributed big data analytics in the edge," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1-36, 2021.

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

[34] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proc. IEEE*, vol. 107, no. 11, pp. 2204-2239, 2019.

[35] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 2, pp. 869-904, 2020.

[36] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849-13875, 2021.

[37] Y. Sun, H. Ochiai, and H. Esaki, "Decentralized deep learning for multi-access edge computing: A survey on communication efficiency and trustworthiness," *IEEE Trans. Artif. Intell.*, vol. 3, no. 6, pp. 963-972, 2021.

[38] W. Su, L. Li, F. Liu, M. He, and X. Liang, "AI on the edge: a comprehensive review," *Artif. Intell. Rev.*, vol. 55, no. 8, pp. 61256183, 2022.

[39] J. Yao et al., "Edge-cloud polarization and collaboration: A comprehensive survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6866-6886, 2023.

[40] S. Duan et al., "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surv. and Tutorials*, vol. 25, no. 1, pp. 591-624, 2023.

[41] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1-30, 2022.

[42] Y. Gong et al., "EdgeRec: Recommender System on Edge in Mobile Taobao," in *Proc. 29th Int. Conf. Inf. Knowl. Manag. (CIKM)*, 2020, pp. 2477-2484.

[43] C. Ding, A. Zhou, Y. Liu, R. Chang, C. H. Hsu, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1489-1499, 2020.

[44] F. Zhuang et al., "A comprehensive survey on transfer learning," *Proc. IEEE*, vol. 109, no. 1, pp. 43-76, 2020.

[45] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 43964415, 2023.

[46] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789-1819, 2021.

[47] L. Wang and K. J. Yoon, "Knowledge distillation and student-teacher Learning for visual intelligence: A review and new outlooks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 6, pp. 30483068, 2022.

[48] C. Yang et al., "Cloud-edge-device collaboration mechanisms of cloud manufacturing for customized and personalized products," in *2022 IEEE 25th Int. Conf. Comput. Support. Coop. Work Des. (CSCWD)*, 2022, pp. 1517-1522.

[49] Y. Cheng, K. Chen, H. Sun, Y. Zhang, and F. Tao, "Data and knowledge mining with big data towards smart production," *J. Ind. Inf. Integr.*, vol. 9, pp. 1-13, 2018.

[50] P. Gupta, Y. Chaudhary, T. Runkler, and H. Schtze, "Neural topic modeling with continual lifelong learning," in *37th Int. Conf. Mach. Learn., (ICML)*, 2020, pp. 38653875.

[51] P. Samira et al., "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 92:1-92:36, 2019.

[52] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 5471, 2019.

[53] Y. S. Liang and W. J. Li, "Adaptive plasticity improvement for continual learning, in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 78167825, 2023.

[54] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Rob. Auton. Syst.*, vol. 15, pp. 2546, 1995.

[55] 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. 55135533, 2023.

[56] D. Kudithipudi et al., "Biological underpinnings for lifelong learning machines," *Nat. Mach. Intell.*, vol. 4, no. 3, pp. 196210, 2022.

[57] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "MLaaS: machine learning as a service," in *2015 IEEE 14th Int. Conf. Mach. Learn. Appl. (ICMLA)*, 2015, pp. 896-902.

[58] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1-36, 2019.

[59] Y. Deng, M. M. Kamani, and M. Mahdavi, "Distributionally robust federated averaging," in *Proc. 34th Adv. Neural Inf. Process. Syst. (NeurIPS), 2020, pp. 1511115122.*

[60] A. Sapio et al., "Scaling distributed machine learning with in-network aggregation," in *Proc. 18th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, 2021, pp. 785-802.

[61] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[62] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018. [Online]. Available: https://arxiv.org/abs/1806.00582.

[63] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, "LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced compu-

tational complexity on IID and non-IID intensive care data," *PLoS One*, vol. 15, no. 4, p. e0230706, 2020.

[64] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-IID private data," 2018. [Online]. Available: http://arxiv.org/abs/1811.11479.

[65] Z. Tang, Y. Zhang, S. Shi, X. He, B. Han, and X. Chu, "Virtual Homogeneity Learning: Defending against Data Heterogeneity in Federated Learning," in *Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 21111-21132.

[66] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 429-450, 2020.

[67] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-IID data quagmire of decentralized machine learning," in *Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 4387-4398.

[68] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," 2019. [Online]. Available: http://arxiv.org/abs/1909.06335.

[69] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Int. Conf. Mach. Learn. (ICML)*, 2020, vol. pp. 5088-5099.

[70] S. Reddi et al., "Adaptive federated optimization," in *Proc. 9th Int. Conf. Learn. Represent. (ICLR)*, 2021.

[71] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-Learning with fast convergence and efficient communication," 2018. [Online]. Available: http://arxiv.org/abs/1802.07876.

[72] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," in *Proc. 34th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020, PP. 3557-3568.

[73] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J. M. Alvarez, "Personalized Federated Learning with First Order Model Optimization," in *Int. Conf. Learn. Represent. (ICLR)* 2020.

[74] D. A. E. Acar et al., "Debiasing model updates for improving personalized federated training," in *Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 21-31.

[75] K. Singhal, H. Sidahmed, Z. Garrett, S. Wu, K. Rush, and S. Prakash, "Federated reconstruction: Partially local federated learning," in *Proc. 35th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2021, vol. 14, pp. 11220-11232.

[76] Y. Jiang, J. Konen, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," 2019. [Online]. Available: http://arxiv.org/abs/1909.12488.

[77] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 34, no. 12, pp. 9587-96032022, 2023.

[78] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. 33th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp.103-112.

[79] A. Harlap et al., "PipeDream: Fast and efficient pipeline parallel DNN training," 2018. [Online]. Available: https://arxiv.org/abs/1806.03377.

[80] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019. [Online]. Available: https://arxiv.org/abs/1909.08053.

[81] Q. Xu, S. Li, C. Gong, and Y. You, "An efficient 2D method for training super-large deep learning models," 2021. [Online]. Available: https://arxiv.org/abs/2104.05343.

[82] B. Wang, Q. Xu, Z. Bian, and Y. You, "2.5-Dimensional distributed model training," 2021. [Online]. Available: https://arxiv.org/abs/2105.14500.

[83] Z. Bian, Q. Xu, B. Wang, and Y. You, "Maximizing parallelism in distributed training for huge neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2105.14450.

[84] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464-2469.

[85] J. Shao, H. Zhang, Y. Mao, and J. Zhang, "Branchy-GNN: A device-edge co-inference framework for efficient point cloud processing," in *IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2021, pp. 8488-8492.

[86] W. Fang, F. Xue, Y. Ding, N. Xiong, and V. C. M. Leung, "EdgeKE: An on-demand deep learning IoT system for cognitive big data on industrial edge devices," *IEEE Trans. Ind. Informatics*, vol. 17, no. 9, pp. 6144-6152, 2021.

[87] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 328-339.

[88] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, and J. Chen, "A lightweight collaborative recognition system with binary convolutional neural network for mobile web augmented reality," in *Proc. 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 1497-1506.

[89] Y. Huang et al., "A lightweight collaborative deep neural network for the mobile web in edge cloud," *IEEE Trans. Mob. Comput.*, vol. 21, no. 7, pp. 2289-2305, 2022.

[90] Y. Huang et al., "An integrated cloud-edge-device adaptive deep learning service for cross-platform web," *IEEE Trans. Mob. Comput.*, vol. 22, no. 4, pp. 1950-1967, 2023.

[91] L. Ren, Y. Liu, X. Wang, J. Lu, and M. J. Deen, "Cloudedge-based lightweight temporal convolutional networks for remaining useful life prediction in IIoT," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12578-12587, 2021.

[92] C. Yang, Z. Lai, Y. Wang, S. Lan, L. Wang, and L. Zhu, "A novel bearing fault diagnosis method based on stacked autoencoder and end-edge collaboration" in *2023 IEEE 26th Int. Conf. Comput. Support. Coop. Work Des. (CSCWD)*, 2023, pp. 393-398.

[93] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: synergistic progressive inference of neural networks over device and cloud," in *Proc. 26th Int. Conf. Mob. Comput. Networking (MobiCom)*, 2020, pp. 488-502.

[94] Z. Fei, X. Yan, S. Wang, and Q. Tian, "Deecap: Dynamic early exiting for efficient image captioning," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp.12216-12226.

[95] X. Li et al., "Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference," *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2023, pp. 8657-8665.

[96] A. Ghodrati, B. E. Bejnordi, and A. Habibian, "Frameexit: Conditional early exiting for efficient video recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 15608-15618.

[97] J. Xin, R. Tang, Y. Yu, and J. Lin, "BERxiT: Early exiting for BERT with better fine-tuning and extension to regression" *Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguist. (EACL)*, 2021, pp. 91-104.

[98] T. Sun et al., "A simple hash-based early exiting approach for language understanding and generation," *Proc. Assoc. Comput. Linguist. (ACL)*, 2022, pp. 2409-2421

[99] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1-9.

[100] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1404.5997.

[101] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615-629, 2017.

[102] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mob. Comput.*, vol. 20, no. 2, pp. 565-576, 2021.

[103] H. Zhou, W. Zhang, C. Wang, X. Ma, and H. Yu, "Bbnet: a novel convolutional neural network structure in edge-cloud collaborative inference," *Sensors*, vol. 21, no. 13, p. 4494, 2021.

[104] N. Shan, Z. Ye, and X. Cui, "Collaborative intelligence: Accelerating deep neural network inference via device-edge synergy," *Secur. Commun. Networks*, vol. 2020, pp. 1-10, 2020.

[105] N. D. Lane et al., "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *15th Int. Conf. Inf. Process. Sens. Networks (IPSN)*, 2016, pp. 23:1-23:12.

[106] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things," *IEEE Netw.*, vol. 33, no. 5, pp. 96-103, 2019.

[107] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. 2018 Work. Mob. Edge Commun. (MECOMM)*, 2018, pp. 31-36.

[108] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 1, pp. 447-457, 2020.

[109] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Des. Autom. & Test Eur. Conf. & Exhib. (DATE)*, 2017, pp. 1396-1401.

[110] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348-2359, 2018.

[111] B. Zhao, Q. Cui, R. Song, Y. Qiu, and J. Liang, "Decoupled Knowledge Distillation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 1194311952.

[112] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 3962-3971.

[113] Y. Chen, N. Wang, and Z. Zhang, "Darkrank: Accelerating deep metric learning via cross sample similarities transfer," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2018, pp. 2852-2859.

[114] F. Tung and G. Mori, Similarity-preserving knowledge distillation" *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1365-1374.

[115] A. Gotmare, N. Shirish Keskar, C. Xiong, and R. Socher, "A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation," in *Int. Conf. Learn. Represent. (ICLR)*, 2019.

[116] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Int. Conf. Learn. Represent. (ICLR)*, 2015.

[117] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *5th Int. Conf. Learn. Represent. (ICLR)*, 2017.

[118] X. Li, H. Xiong, H. Wang, Y. Rao, L. Liu, and J. Huan, "Delta: Deep learning transfer using feature map with attention for convolutional networks," in *Int. Conf. Learn. Represent. (ICLR)*, 2019.

[119] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proc. 30th IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 7130-7138.

[120] S. Srinivas and F. Fleuret, "knowledge transfer with jacobian matching,"' in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4730-4738.

[121] S. Lee and B.C. Song, "Graph-based knowledge distillation by multi-head attention network" in *Proc. 30th Br. Mach. Vis. Conf. (BMVC)*, 2019, p.141.

[122] N. Passalis, M. Tzelepi and A. Tefas, "Heterogeneous knowledge distillation using information flow modeling," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 2336-2345.

[123] Y. Lu et al., "Collaborative learning between cloud and end devices: An empirical study on location prediction," in *Proc. 4th Symp. Edge Comput. (SEC)*, 2019, pp. 139-151.

[124] T. Jing, X. Tian, H. Hu, and L. Ma, "Cloud-edge collaboration framework with deep learning-based for remaining useful life prediction of machinery," *IEEE Trans. Ind. Informatics*, vol. 18, no. 10, pp. 7208-7218, 2022.

[125] J. Yoon, W. Jeong, G. Lee, E. Yang, and S. J. Hwang, "Federated continual learning with weighted inter-client transfer," in *Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 12073-12086.

[126] J. Dong et al., "Federated class-incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 10164-10173.

[127] J. Le, X. Lei, N. Mu, H. Zhang, K. Zeng, and X. Liao, "Federated continuous learning with broad network architecture," *IEEE Trans. Cybern.*, vol. 51, no. 8, pp. 3874-3888, 2021.

[128] Y. Ma, Z. Xie, J. Wang, K. Chen, and L. Shou, "Continual federated learning based on knowledge distillation," in *Proc. Thirty-First Int. Jt. Conf. Artif. Intell. (IJCAI)*, 2022, pp. 2182-2188.

[129] Y. Luopan, R. Han, Q. Zhang, C. H. Liu, G. Wang, and L. Y. Chen, Fedknow: Federated continual learning with signature task knowledge integration at edge, in *2023 IEEE 39th Int. Conf. Data Eng. (ICDE)*, 2023, pp. 341-354.

[130] Z. Wang, Y. Zhang, X. Xu, Z. Fu, H. Yang, and W. Du, "Federated probability memory recall for federated continual learning," *Inf. Sci. (Ny).*, vol. 629, pp. 551565, 2023.

[131] D. Shenaj, M. Toldo, A. Rigon, and P. Zanuttigh, "Asynchronous federated continual learning," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2023. pp. 5055-5063.

[132] Z. Zhang, Y. Zhang, D. Guo, S. Zhao, and X. Zhu, Communication-efficient federated continual learning for distributed learning system with Non-IID data, *Sci. China Inf. Sci.,* vol. 66, no. 2, p. 122102, 2023.

[133] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," 2019. [Online] Available: https://arxiv.org/abs/1902.09574

[134] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 129-146, 2020.

[135] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1-124, 2021.

[136] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A Comprehensive Survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485-532, 2020.

[137] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929-1958, 2014.

[138] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *Int. Conf. Learn. Represent. (ICLR)*, 2019.

[139] T. Chen et al., "The lottery ticket hypothesis for pre-trained BERT networks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 15834-15846.

[140] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 3592-3602.

[141] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?" in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 11890-11899.

[142] H. Wang, C. Qin, Y. Bai, Y. Zhang, and Y. Fu, "Recent advances on neural network pruning at initialization," in *Proc. Int. Jt. Conf. Artif. Intell. (IJCAI)*, 2022, pp. 5638-5645.

[143] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014. [Online] Available: https://arxiv.org/abs/1412.6115.

[144] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2018, pp.4335-4342.

[145] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2285-2294.

[146] M. Courbariaux, Y. Bengio, and J. P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2015, pp. 3123-3131.

[147] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks," in *Int. Conf. Embed. Wirel. Syst. Networks (EWSN)*, 2017, pp. 168-173.

[148] H. Kim, K. Kim, J. Kim, and J.-J. Kim, "BinaryDuo: Reducing gradient mismatch in binary activation network by coupling binary activations," in *Int. Conf. Learn. Represent. (ICLR)*, 2020.

[149] U. Kster et al., "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 1743-1753.

[150] N. Wang, J. Choi, D. Brand, C. Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 7675-7684.

[151] S. Khoram and J. Li, "Adaptive quantization of neural networks," in *Int. Conf. Learn. Represent. (ICLR)*, 2018.

[152] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 365-382.

[153] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 8612-8620.

[154] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 2146-2156.

[155] Z. Yu and Y. Shi, "Kernel quantization for efficient network compression," *IEEE Access*, vol. 10, pp. 4063-4071, 2022.

[156] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," *Int. Conf. Learn. Represent. (ICLR)* 2017.

[157] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size," 2016. [Online] Available: https://arxiv.org/abs/1602.07360.

[158] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1251-1258.

[159] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510-4520.

[160] A. Howard et al., "Searching for mobilenetv3," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1314-1324.

[161] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *Int. Conf. Learn. Represent. (ICLR)*, 2017.

[162] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *Int. Conf. Learn. Represent. (ICLR)*, 2019.

[163] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1304-1313.

[164] M. Tan et al., "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 2820-2828.

[165] B. Wu et al., "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 10734-10742.

[166] X. Dai et al., "ChamNet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 11398-11407.

[167] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 10691-10700.

[168] A. Wan et al., "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 12965-12974.

[169] H. Chen et al., "Binarized neural architecture search for efficient object recognition," *Int. J. Comput. Vis.*, vol. 129, no. 2, pp. 501-516, 2021.

[170] H. Chen et al., "Binarized neural architecture search," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI)*, 2020, pp. 10526-10533.

[171] M. Shen, K. Han, C. Xu, and Y. Wang, "Searching for accurate binary neural architectures," in *Proc. Int. Conf. Comput. Vis., (ICCV)*, 2019, pp. 2041-2044.

[172] B. Lyu, S. Wen, K. Shi, and T. Huang, "Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 1158-1169, 2023.

[173] P. Ren et al., "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1-34, 2021.

[174] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1423-1431.

[175] S. Zhang et al., "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1-24, 2020.

[176] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *IEEE 24th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2018, pp. 671-678.

[177] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: Dnn model slicing for visual analytics containers at the edge," in *Proc. 4th Symp. Edge Comput. (SEC)*, 2019, pp. 179-194.

[178] L. Hu, G. Sun, and Y. Ren, "CoEdge: Exploiting the edge-cloud collaboration for faster deep learning," *IEEE Access*, vol. 8, pp. 100533-100541, 2020.

[179] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *Proc. 15th IEEE Int. Conf. Adv. Video Signal-Based Surveill. (AVSS)*, 2019, pp. 1-6.

[180] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proc. 4th Symp. Edge Comput. (SEC)*, 2019, pp. 195-208.

[181] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, " DeeperThings: Fully distributed CNN inference on resource-constrained edge devices," *Int. J. Parallel Program.*, vol. 49, pp. 600-624, 2021.

[182] L. Zheng, G. Liu, C. Yan, C. Jiang, M. Zhou, and M. Li, "Improved TrAdaBoost and its application to transaction fraud detection," *IEEE Trans. Comput. Soc. Syst.*, vol. 7, no. 5, pp. 1304-1316, 2020.

[183] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345-1359, 2010.

[184] Z. Zhu, K. Lin, A. K. Jain and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2023. DOI: 10.1109/TPAMI.2023.3292075.

[185] C. Bucil, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, 2006, pp. 535-541.

[186] T. Fukuda, M. Suzuki, G. Kurata, S. Thomas, J. Cui, and B. Ramabhadran, "Efficient knowledge distillation from an ensemble of teachers," in *Proc. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2017, pp. 3697-3701.

[187] A. Wu, W. S. Zheng, X. Guo, and J. H. Lai, "Distilled person re-identification: Towards a more scalable system," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 1187-1196.

[188] L. Jiang et al., "Long short-term sample distillation," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI)*, 2020, pp. 4345-4352.

[189] X. Jin, C. Lan, W. Zeng, and Z. Chen, "Uncertainty-aware multi-shot knowledge distillation for image-based object re-identification," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI)*, 2020, pp. 11165-11172.

[190] C. Shen, M. Xue, X. Wang, J. Song, L. Sun, and M. Song, "Customizing student networks from heterogeneous teachers via adaptive knowledge amalgamation," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 3504-3513.

[191] C. Shen, X. Wang, J. Song, L. Sun, and M. Song, "Amalgamating knowledge towards comprehensive classification," in *Proc. 33th AAAI Conf. Artif. Intell. (AAAI)*, 2019, pp. 3068-3075.

[192] D. Xu, W. Ouyang, X. Wang, and N. Sebe, "Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 675-684.

[193] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI)*, 2020, pp. 5191-5198.

[194] S. Albanie, A. Nagrani, A. Vedaldi, and A. Zisserman, "Emotion recognition in speech using cross-modal transfer in the wild," in *Proc. 26th ACM Int. Conf. Multimed.(ACM MM*, 2018, pp. 292-301.

[195] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4320-4328.

[196] D. Chen, J.-P. Mei, C. Wang, Y. Feng, and C. Chen, "Online knowledge distillation with diverse peers," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI)*, 2020, vol. 34, no. 04, pp. 3430-3437.

[197] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," *Proc. 6th Int. Conf. Learn. Represent. (ICLR)* 2018.

[198] Z. Li et al. "When object detection meets knowledge distillation: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 8, pp. 10555-10579, 2023.

[199] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nat. Commun.*, vol. 13, no. 1, p. 2032, 2022.

[200] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto, "Distillation-based semi-supervised federated learning for communication-efficient collaborative training with Non-IID private data," *IEEE Trans. Mob. Comput.*, vol. 22, no. 1, pp. 191-205, 2023.

[201] F. Sattler, A. Marban, R. Rischke, and W. Samek, "CFD: Communication-efficient federated distillation via soft-label quantization and delta coding," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2025-2038, 2021.

[202] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 14068-14080, 2020.

[203] S. Cheng, J. Wu, Y. Xiao, Y. Liu, and Y. Liu, "FedGEMS: Federated learning of larger server models via selective knowledge fusion," 2021, [Online]. Available: http://arxiv.org/abs/2110.11027.

[204] Y. J. Cho, A. Manoel, G. Joshi, R. Sim, and D. Dimitriadis, "Heterogeneous ensemble knowledge transfer for training large models in federated learning, in *Int. Jt. Conf. Artif. Intell. (IJCAI)*, 2022, pp. 2881-2887.

[205] S. Yu, W. Qian, and A. Jannesari, "Resource-aware federated learning using knowledge extraction and multi-model fusion," 2022, [Online]. Available: http://arxiv.org/abs/2208.07978.

[206] L. Zhang, D. Wu, and X. Yuan, "FedZKT: Zero-shot knowledge transfer towards resource-constrained federated learning with heterogeneous on-device models," in *Proc. Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2022, pp. 928-938.

[207] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, pp. 2351-2363, 2020.

[208] Z. Zhu, J. Hong, and J. Zhou, "Data-free knowledge distillation for heterogeneous federated learning," in *Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 12878-12889.

[209] L. Zhang, L. Shen, L. Ding, D. Tao, and L.-Y. Duan, "Fine-tuning global model via data-free knowledge distillation for non-iid federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 10174-10183.

[210] H. Wang, Y. Li, W. Xu, R. Li, Y. Zhan, and Z. Zeng, "DaFKD: Domain-aware federated knowledge distillation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2023, pp. 20412-20421.

[211] D. Yao et al., "FedGKD: Towards heterogeneous federated learning via global knowledge Distillation," *IEEE Trans. Comput.*, 2023.

[212] D. Sui, Y. Chen, J. Zhao, Y. Jia, Y. Xie, and W. Sun, "Feded: Federated learning via ensemble distillation for medical relation extraction," in *Proc. 2020 Conf. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2020, pp. 2118-2128.

[213] Z. Wu et al., "FedICT: Federated multi-task distillation for multi-access edge computing," *IEEE Trans. Parallel Distrib. Syst.*, 2023.

[214] H. Jin et al., "Personalized edge intelligence via federated self-knowledge distillation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 567-580, 2023.

[215] X. Gong et al., "Preserving privacy in federated learning with ensemble cross-domain knowledge distillation," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2022, pp. 11891-11899.

[216] R. Bommasani et al., "On the opportunities and risks of foundation models," 2021, [Online]. Available: http://arxiv.org/abs/2108.07258

[217] C. Zhou et al., "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," 2023, [Online]. Available: https://arxiv.org/abs/2302.09419

[218] N. Ding et al., "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nat. Mach. Intell.*, vol. 5, no. 3, pp. 220-235, 2023.

[219] E. Ben Zaken, S. Ravfogel, and Y. Goldberg, "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," in *Proc. 60th Annu. Meet. Assoc. Comput. Linguist. (ACL)*, 2022, pp.1-9.

[220] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 4944-4953.

[221] R. He et al., On the effectiveness of adapter-based tuning for pretrained language model adaptation, in *Proc. 59th Annu. Meet. Assoc. Comput. Linguist. 11th Int. Jt. Conf. Nat. Lang. Process. (ACL-IJCNLP)*,2021, pp. 2208-2222.

[222] Y.-L. Sung, J. Cho, and M. Bansal, "Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 5227-5237.

[223] Y. Wang et al., "AdaMix: Mixture-of-adaptations for parameter-efficient model tuning," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2022, pp.5744-5760.

[224] A. Rckl et al., "Adapterdrop: On the efficiency of adapters in transformers," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2021, pp.7930-7946.

[225] S. He, L. Ding, D. Dong, M. Zhang, and D. Tao, "SparseAdapter: An easy approach for improving the parameter-efficiency of adapters," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2022, pp. 2184-2190.

[226] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2021, pp. 3045-3059.

[227] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proc. 59th Annu. Meet. Assoc. Comput. Linguist. 11th Int. Jt. Conf. Nat. Lang. Process. (ACL-IJCNLP)* 2021, pp. 4582-4597.

[228] R. Zhang et al., "LLaMA-adapter: Efficient fine-tuning of language models with zero-init attention," 2023, [Online]. Available: http://arxiv.org/abs/2303.16199

[229] X. Liu et al., "P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks," in *Proc. 60th Annu. Meet. Assoc. Comput. Linguist. (ACL)*, 2022, pp. 6168.

[230] X. Liu et al., "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," 2021, [Online]. Available: https://arxiv.org/abs/2110.07602

[231] B. Zhu, Y. Niu, Y. Han, Y. Wu, and H. Zhang, "Prompt-aligned gradient for prompt tuning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)* 2022, pp. 15659-15669.

[232] Y. Gu, X. Han, Z. Liu, and M. Huang, "PPT: Pre-trained prompt tuning for few-shot learning," in *Proc. 60th Annu. Meet. Assoc. Comput. Linguist. (ACL)*, 2022, PP. 8410-8423.

[233] Z. Wang, R. Panda, L. Karlinsky, R. Feris, H. Sun, and Y. Kim, "Multitask prompt tuning enables parameter-efficient transfer learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[234] E. Hu et al., "Lora: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[235] V. Lialin, V. Deshpande, and A. Rumshisky, "Scaling down to scale up: A guide to parameter-efficient fine-tuning," 2023, [Online]. Available: https://arxiv.org/abs/2303.15647

[236] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, "LoRA-FA: Memory-efficient low-rank adaptation for large language models Fine-tuning," 2023, [Online]. Available: https://arxiv.org/abs/2308.03303

[237] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs, 2023, [Online]. Available: http://arxiv.org/abs/2305.14314

[238] Y. Xu et al., "QA-LoRA: Quantization-aware low-rank adaptation of large language models," 2023, [Online]. Available: http://arxiv.org/abs/2309.14717

[239] J. Kim et al., "Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization," 2023, [Online]. Available: http://arxiv.org/abs/2305.14152

[240] Y. Li et al., "LoftQ: LoRA-fine-tuning-aware quantization for large language models," 2023, [Online]. Available: http://arxiv.org/abs/2310.08659

[241] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu, "Full parameter fine-tuning for large language models with limited resources," 2023, [Online]. Available: http://arxiv.org/abs/2306.09782

[242] S. Malladi et al., "Fine-tuning language models with just forward passes," 2023, [Online]. Available: http://arxiv.org/abs/2305.17333

[243] H. He, J. Cai, J. Zhang, D. Tao, and B. Zhuang, Sensitivity-aware visual parameter-efficient fine-tuning, in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2023, pp. 11825-11835.

[244] X. Tang et al., "Computing power network: The architecture of convergence of computing and networking towards 6G requirement," *China Commun.*, vol. 18, no. 2, pp. 175-185, 2021.

[245] C. Yang, Y. Wang, Y. Jiang, S. Lan, and L. Wang, "Metaverse: Architecture, technologies, and industrial applications," in *2023 IEEE 19th Int. Conf. Autom. Sci. Eng. (CASE)*, 2023, pp. 16.

[246] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7436-7456, 2022.

[247] Y. Zhong, J. Zhou, P. Li, and J. Gong, "Dynamically evolving deep neural networks with continuous online learning," *Inf. Sci.*, vol. 646, p. 119411, 2023.

[248] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The Principles, Progress, and Challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126136, 2018.

[249] H. H. Zhu et al., "Space-efficient optical computing with an integrated chip diffractive neural network," *Nat. Commun.*, vol. 13, no. 1, p. 1044, 2022.

[250] T. Wang et al., "Image sensing with multilayer nonlinear optical neural networks," *Nat. Photonics*, vol. 17, no. 5, pp. 408-415, 2023.

[251] C. Yang, S. Lan, Z. Zhao, M. Zhang, W. Wu, and G. Q. Huang, "Edge-Cloud blockchain and IoE-enabled quality management platform for perishable supply chain logistics," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3264-3275, 2023.

[252] M. Ayyat, S.K. Nukavarapu, T. Nadeem, "Dynamic deep neural network adversarial attacks for edge-based IoT devices," *Proc. 2022 IEEE Glob. Commun. Conf. (GLOBECOM)*, 2022, pp.61-67.