Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.023684 Published Online: 02 November 2022

check for updates

ARTICLE

Edge Intelligence with Distributed Processing of DNNs: A Survey

Sizhe Tang[1], **Mengmeng Cui**[1,*], **Lianyong Qi**[2] **and Xiaolong Xu**[1]

[1]School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210000, China

[2]School of Computer Science, Qufu Normal University, Qufu, 273100, China

*Corresponding Author: Mengmeng Cui. Email: cuimengmeng@nuist.edu.cn

Received: 09 May 2022 Accepted: 16 September 2022

ABSTRACT

With the rapid development of deep learning, the size of data sets and deep neural networks (DNNs) models are also booming. As a result, the intolerable long time for models' training or inference with conventional strategies can not meet the satisfaction of modern tasks gradually. Moreover, devices stay idle in the scenario of edge computing (EC), which presents a waste of resources since they can share the pressure of the busy devices but they do not. To address the problem, the strategy leveraging distributed processing has been applied to load computation tasks from a single processor to a group of devices, which results in the acceleration of training or inference of DNN models and promotes the high utilization of devices in edge computing. Compared with existing papers, this paper presents an enlightening and novel review of applying distributed processing with data and model parallelism to improve deep learning tasks in edge computing. Considering the practicalities, commonly used lightweight models in a distributed system are introduced as well. As the key technique, the parallel strategy will be described in detail. Then some typical applications of distributed processing will be analyzed. Finally, the challenges of distributed processing with edge computing will be described.

KEYWORDS

Distributed processing; edge computing; parallel strategies; acceleration of DNN processing

1 Introduction

In recent years, the rapid development and popularity of deep learning have promoted the progress of various fields [1–3], including intelligent medicine, automated driving, smart home and so on. DNNs [4], the core components of deep learning, are used to complete the tasks such as image classification and natural language processing by extracting the intrinsic features of the input to help people achieve pattern recognition and decision making. The cost of deep learning lies mainly in two aspects: the time consumed for the training as well as inference of neural networks and the requirement of materials like hardware providing computing power or the recourse of devices. Nowadays, to get the more accurate results of prediction or decision, the volume of data sets and the size of models are booming as well [5–7], resulting the intolerable time consumption. However, some devices can not afford the size of models or recourse required by the processing. As a result, the distributed scheme [8–10] where multiple devices cooperate to complete the computing tasks has become an inevitable trend since the single performance of common devices could not deal with these hard tasks well. Moreover, since the distributed processing usually takes place in a parallel way, the efficiency of training and inference of large DNNs models also gains a significant increase.

In the conventional way of distributed processing, tasks are allocated or offloaded to different devices simply, considering little about the fine-grained distribution. In the era of the Internet of Things (IoT), massive devices organized under the paradigm of edge computing have greater performance and potential in realizing distributed processing compared with the conventional way [11,12]. In the scenario of edge computing [13], there are closer ties between edge devices as well as edge servers so that they can execute assigned parallel strategies well. From another point of view, the distribution of devices also helps improve the edge computing system. Due to the characteristics of modern large-scale DNNs models, the models could be partitioned into different edge devices or different blocks in one device so to make the sub-tasks be completed parallelly. Data sets can also be partitioned into subsets and assigned to different devices deployed with the same model. Moreover, there is a further parallel strategy called hybrid parallelism [14] applied in the distribution, which combines the advantages of both model and data parallelism. Therefore, the original large models or data sets can be processed efficiently. In general, the distribution with parallelism in edge computing has the following contributions to accelerating the processing of large-scale DNN models:

- The efficiency of processing is improved significantly.
- The utilization of devices or hardware is higher with suitable parallel schemes.
- The useless communication between devices in edge computing can be reduced during processing.

At present, there are various frameworks, languages and applications based on the strategies mentioned above being studied. To offer systematical elaboration of the strategy for accelerating the processing of DNNs models leveraging the ideas of distributed operation and parallel schemes in edge computing, a concrete and enlightening survey of the recent efforts is conducted in this literature. Additionally, considering the practical demand for recourse-constrained devices in edge computing, several popular lightweight models are introduced for the sake of providing new ideas for conducting distributed processing better in the world of IoT. In this paper, we integrate and propose some methods and ideas for optimizing deep learning in edge computing using distributed processing, especially focusing on parallel processing.

Although there have been many papers on distributed deep learning and federated learning, this paper has its significance to exist. These existing papers usually focus on a single perspective and do not explain how distributed processing of DNN is facilitated in edge computing. Besides, papers on federated learning do not provide a comprehensive survey on distributed processing of DNN in edge computing scenarios. Moreover, distributed deep learning is only a strategy to accelerate the processing (training and inference) of deep learning and federated learning aims at protecting users' privacy and helping corporate to model while edge intelligence's goal is to promote the collaboration of multiple devices for the optimization of the whole edge system. As a result, this paper provides a survey of edge computing combined with deep learning in a distributed way from a higher and more holistic perspective, with the expectation that it will inspire more novel research on these aspects.

The organization of the rest part is as follows. Section 2 introduces the related background of DNN and edge computing and the latter is presented in detail to inspire more ideas about combining edge computing and distributed processing. In Section 3, some lightweight models are introduced since they are novel in implementing distribution. The key technique of parallelism is presented in Section 4.

Representative applications based on distribution, parallelism and edge computing will be presented and analyzed in detail in Section 5 for inspiring further research. In Section 6, challenges and future directions are given. Finally, the paper will be concluded in Section 7.

2 Preliminary of DNN and EC

2.1 DNN

Inspired by the framework of animals' visual systems, DNNs have been created with multiple layers stacked to extract the features from the input data. Recently, DNNs have become an important method in machine learning or deep learning since it has a great ability to find the distributed expression of data. In this subsection, some typical DNN models [15] will be listed and a most popular method for training will be introduced. Fig. 1 shows some fields where DNNs could be made used. Since the following models could be seen as the basic and core components for DNN models, the general methods to combine complicated DNN models consisting of them are analyzed in the third sub-section.

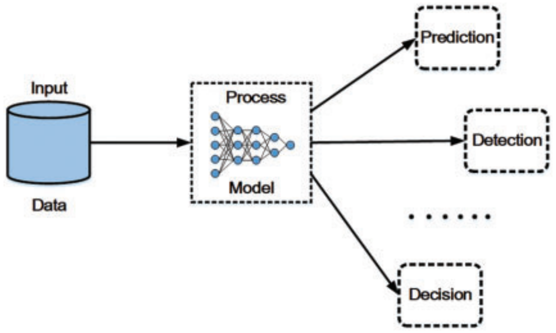

Figure 1: Areas where DNNs could be applied

Deep Belief Network A Deep Belief Network (DBN) is made up of a large number of Restricted Boltzmann Machines (RBMs), which is a two-layer network for unsupervised learning and can fit the samples' feature [16]. Since RBMs are stacked in a sequence in a DBN, the former RBM's output can be viewed as the latter's input. This process presents how the DBN continuously extracts further features from the extracted feature. Based on this thought, more and more large models with huge numbers of layers are created for deep learning to achieve better classification tasks.

Stacked Autoencoder The basic components of an autoencoder are the encoder and the decoder, which are both made up of numbers of neurons. The input data in the form of vectors are compressed by the hidden layer and then the input vector becomes a vector of lower dimension, namely encoding. The decoder is required to map the vector produced by the hidden layer back to the original input space with the mapping functions. Then we will optimize the autoencoder by minimizing the average error between the input data with the coded data. After lots of rounds of training, test data can be fed into it to gain the result by comparing the error with the threshold. Additionally, activation functions of encoders and decoders are supposed to be non-linear functions, which can extract the non-linear correlation of input data's features. Different from the conventional autoencoder, the convolutional autoencoder is proposed in paper [17]. By stacking these autoencoders in a sequence, a DNN model is constructed and some more complex tasks can be tackled.

Deep Convolutional Neural Network In recent years, the Convolutional Neural Network (CNN) has attracted great interest from researchers in various fields, especially in pattern recognition, whose name comes from a linear operation in mathematics called convolution. As a multiple-layer network, CNN consists of four components: layers for convolution, non-linearity, pooling and full connection [18]. CNNs perform significantly well especially in tasks dealing with image data such as image classification and natural language processing (NLP).

The core operation of CNN is convolution. The diagram of this operation is shown in Fig. 2. Abstractly speaking, in convolution, the input image is partitioned into numbers of regions with exact size (decided by the width and height) and then these regions are connected to the next layer's neurons correspondingly. That is to say, every neuron only receives data from the corresponding region. Due to the convolution, the input data's dimension is dropped dramatically and the features are extracted at the same time. Similar to other neural networks, non-linearity functions are applied in neurons as well. Pooling layers are used in CNNs to reduce the complexity of computation for the following layers. For example, max-pooling partitions the sub-regions into further rectangles and uses their maximum value to represent the sub-region. The idea can be summarized as down-sampling. The fully-connected layers have a similar mechanism and function as conventional neural networks. Additionally, the idea of CNN is also learned by other neural networks.

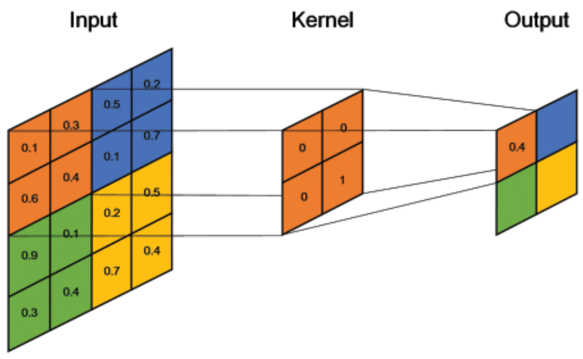

Figure 2: The diagram of convolution

Recurrent Neural Network In addition to networks consisting of sequential layers, many networks are organized circularly. Recurrent Neural Network (RNN) [19] is a typical architecture of networks in deep learning. RNN is good at processing sequential information, such as natural language processing (NLP). Its recurrent structure makes it could make use of previous computation in the form of 'memory'. However, vanishing gradients and exploding gradients are significant drawbacks of standard RNNs. To address these issues, LSTM (Long Short-Term Memory) is proposed, which consists of three basic gates: forget gate, input gate and output gate. Forget gate is used to control the influence of historical information on the current memory unit so that the important information is preserved for better processing. Many networks based on RNN or LSTM are applied in text generation, machine translation, voice recognition, image description and so on. The combination of RNN and edge computing is of great practical significance. For example, a deep RNN-based network is applied to classify applications from traffic patterns in the hybrid scenario of edge computing and cloud computing [20]. Some other networks with complicated architecture are also mostly based on the RNN, such as echo state networks (ESNs), and liquid state machines (LSMs).

Generative Adversarial Network Based on game theory, a generative adversarial network (GAN) [21] consisting of two core networks–efficient and distributed training. For exagenerator and discriminator is widely applied in image generation, semantic segmentation and other fields [22]. The generator is supposed to generate fake data by learning the distribution of true data and the discriminator shall classify the fake data from true data. Both of them tend to improve their ability in adversarial processing, which will finish when a Nash equilibrium is found. Through the learned features, the generator can fabricate fake information without being detected.

Training Approaches In the training of DNNs models, the main purpose is to minimize the loss function to optimize the networks, which reflects the gap between the prediction and the true value of samples. There are many methods for optimizing the the training processing of deep learning, such as SGD [23], Momentum [24], NAG [25], Adagrad [26], Adam [27], AMSGrad [28] and so on. Stochastic gradient descent (SGD) is widely applied in the training as a general approach and it is also suitable for distributed training. Unfortunately, since the traditional SGD is inherently sequential, it is hard to train a huge amount of data sets by it because of the long time required by data to complete the trip through the whole model. So, considering the massive data produced by various devices in edge computing or IoT, some improved approaches based on SGD have been proposed to realize efficient and distributed training. For example, De et al. [29] proposed a method for highly parallel platforms and distributed computation. In addition, Downpour SGD [30] was proposed to deal with large data sets in an asynchronous way supporting numbers of model replicas. However, when the surface curvature in one dimension is much larger than in another which usually occurs near the optimal point, SGD is hard to go through the ravine. Momentum is a method that helps SGD accelerate in a related direction and suppress the jittering of gradients. In Momentum, the update of parameters not only relies on the current gradients but also depends on the direction of the parameters' last update. NAG (Nesterov accelerated gradient) is a bit different from Momentum. In NAG, the gradients are computed according to the situation after taking the planned step instead of the current gradients. That is to say, NAG could 'look forward' by using the information of the second derivative. Adagrad is an optimization algorithm based on gradients. In Adagrad, parameters of low frequency are updated with larger step sizes while parameters of high frequency are updated with a smaller step size. By Adagrad, SGD gets more robustness. Adam (Adaptive Moment Estimation) is also an algorithm that could adapt the learning rate. It assigns a learning rate to each parameter. In some scenarios, some mini-batches generate gradients of rich information while they rarely appear. As a result, exponential averaging diminishes their impact so that the convergence is not so well. AMSGrad uses the maximum value of the historical gradient squared to update the parameters instead of using exponential averaging in order to correct the above behavior. There are also some effective methods for optimizing the training of deep learning, such as QHAdam, AGGMo, etc.

Since almost all DNNs could be classified into two categories according to the structure: sequential networks and recurrent networks. The former is easy to be partitioned and then deployed to multiple networks so that the distribution of these models is available. As to the recurrent one, such as some models in the form of a graph, there are also methods to partition them for parallel processing [31].

2.2 EC

In recent years, with the fast development of IoT, millions of devices and sensors are playing various and important roles in our life, such as real-time information collection and monitoring, image transmission, and dynamic solutions to complex problems. At the same time, they are constantly generating huge amounts of data, which gradually creates problems such as resource congestion, as evidenced by high service response delays and information transmission speed hindrance. In such a context, edge computing has become a new paradigm to solve the local computing of IoT devices. Cloud computing reduces the computational pressure on local devices by offloading the computation of local devices directly to the central cloud server, which improves the user experience to a certain extent, but there are still problems such as long transmission time and resource congestion [32]. Compared with traditional cloud computing, edge computing migrates the computation and storage of data from local devices to edge networks, making them closer to users. There are nodes for computation distributed in the edge network, and devices can offload computation to these distributed nodes instead of all to the cloud processing center. Such processing not only reduces the transmission latency between user devices and edge servers but also strongly eases network congestion. In addition, with the advantage of its multi-node architecture, edge computing can achieve system-level optimization by shifting the computation and communication overhead from energy-limited nodes to energy-rich and more capable nodes, i.e., achieving a dynamic distribution of tasks.

Edge intelligence [33] is defined as a set of connected systems and devices that collaborate for data collection, caching, processing and analysis proximity to where data are captured based on artificial intelligence [34]. Empowered by distributed deep learning, edge intelligence is able to make better decisions on how to offload computation and how to allocate sources and so on. That is to say, distributed deep learning promotes edge intelligence to achieve a system with greater scope, which could process assigned tasks with higher efficiency and quality.

2.2.1 Architecture

Different from the architecture of traditional cloud computing, edge computing is aimed to reduce the distance between data and the place where it is computed. The changes are requested by the low latency required by users' modern applications. From the view of hardware, there are three types of devices in an edge computing system respectively for generating raw data, performing data processing, and receiving the processed data. The architecture of edge computing can be hierarchically divided into three layers shown below [35,36]. The architecture of edge computing is shown in Fig. 3.

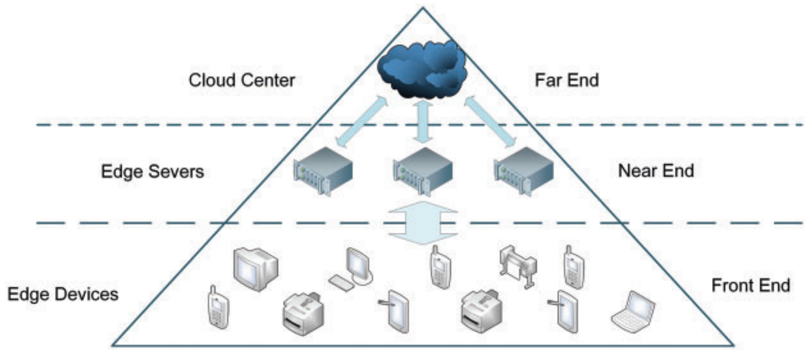

Figure 3: The architecture of edge computing

Front-End In the architecture of edge computing, edge devices compose the front-end, which makes the interaction with end users more frequent and the responsive latency lower. Since nearly all edge devices are able to compute, edge computing could meet the requirements of some real-time applications such as information forwarding. However, due to the limited computing and storage capacity as well as energy, edge devices cannot satisfy most applications' requirements, which need massive computation and consume the life of batteries. Therefore, edge devices need to forward the requirements to edge servers.

Near-End In most cases, the data computation and storage of applications will be migrated to edge servers, which make up the near-end environment. Therefore, the end users could get access to a more enjoyable service through the improvement of data computation and storage as well as transmission latency. Edge servers, also called cloudlets, could be viewed as the miniaturized versions of central cloud servers and realize the distribution of tasks in edge computing as well. They are commonly composed of CPUs and GPUs with one or more processors [35]. Compared to traditional cloud computing, the near-end environment makes servers much closer to users so that the distribution of computation and storage is easier to achieve. However, these servers are not good at providing consistent performance for dynamic streaming data from IO channels. As a result, a general-purpose computing system stack focusing on processing streaming data with higher energy efficiency and lower consumption in the scenario mentioned above [37] is expected to be built. Moreover, edge servers can also meet some requirements including data caching, real-time processing and so on.

Far-End Since cloud servers are deployed in the far-end environment, the transmission latency and consumption are much higher compared with that of edge servers. Nonetheless, due to limitations in computing power as well as storage on edge servers, some of the computation offloaded past by edge devices still can not be done in the near-end environment, so the existence of cloud servers makes a lot of sense. With the powerful computing power and large storage capacity, cloud servers make up for the shortage of edge servers, though tasks being achieved in the near-end environment is an ideal state. As the cloud servers and edge servers are both of significance and have their advantages respectively, their collaboration has become a trend in the related works of edge computing [38]. In recent years, more and more research has revolved around improving the overall performance of the system through the collaboration of them [39,40].

The architecture of edge computing makes the collaboration of edge devices as well as servers more convenient. For example, devices could communicate with each other and exchange data or parameters due to the end-to-end structure; the cloud server could help edge devices and end devices help end devices so that some communication and transportation overhead is removed due to the hierarchical model.

2.2.2 Implements

In order to implement the architecture of edge computing mentioned above, many researches have focused on the designing of edge computing models. There are two models as below that dominate [32].

Hierarchical Model Given that edge servers can be deployed at different locations from edge devices, the entire edge computing architecture can be divided into different tiers. The layers are differentiated by function and distance. So, a hierarchical model can describe the architecture clearly.

Composing the first tier of the structure, the edge servers could receive workloads from the edge devices such as mobile phones and onboard computers directly [41]. Then, these edge servers are connected to cloud servers and remote processing centers that belong to higher tiers through the backbone network.

Due to the hierarchical structure, servers on different tiers can share real-time information on workloads so that they can work cooperatively. That is, if some servers can not afford the workloads of computation or storage, they can further offload them to servers on higher tiers. Consequently, the peak loads can be much larger with the same number of servers in this way.

Software Defined Networking Model Considering the massive applications and sensors on the edge, it is of great complexity to manage edge computing. Software Defined Networking (SDN) is a solution to the problem [42]. Different from the conventional paradigm where networks depend on devices realized by vendors, SDN sets a customized SDN controller that could control the whole network in a logically-centralized way. The new architecture can simplify the management of edge computing and improve the flexibility of the network [42].

SDN is built to separate the data-plane and the control-plane. In this architecture, the network nodes implement the data-plane alone, while the control-plane is realized by a separated architecture element–SDN controller. That is, the SDN controller is software for logical controlling. Therefore, the SDN is capable of easing the design and management of the network by customizing and centralizing the control-plane.

2.2.3 Applications

With the growing development and popularity of edge computing, more and more scenarios are undergoing unprecedented changes with its support, not only in technology but also in people's lifestyles. In this part, several applications enabled by deep learning in edge computing will be introduced.

Smart Multimedia Over the past few years, there has been an increasing demand for multimedia information on the Internet. To meet this demand, better techniques for video processing have become the center point of research in various efforts [43–45], specifically including video processing, caching and delivery. Combining deep learning with edge computing is an effective intelligent processing method.

In terms of video analytics, DeepCham [46], an edge master server that coordinates with participating users to train CNN models for better mobile object recognition. Wang et al. [47] designed a framework assisted by edge computing, which can assign users to the most proper edge servers intelligently by deep reinforcement learning to solve the problems of latency and congestion. Zhang et al. [48] proposed an algorithm named LSTM-C that could decide the cache placement without data pre-processing or additional information.

Smart Transportation Various applications of the Internet of Vehicles (IoV) have emerged to improve the efficiency, security and comfort of driving [49,50]. In the real world, a large number of in-vehicle applications are advanced latency-sensitive, yet edge computing can make the information processing and transmission latency significantly reduced, so the integration of edge computing is an inevitable trend.

In the scenario of autonomous driving, Chen et al. [51] proposed a multi-view 3D deep learning network for high-accuracy 3D objection detection. Lv et al. [52] leveraged deep neural networks to realize the analysis and prediction of traffic by mining the short-term information of traffic situation. Moreover, Chu et al. [53] contributed to the intelligent control according to the traffic features by taking a multi-agent actor-critic approach.

Smart City Edge computing is a powerful distributed paradigm for processing the distributed big data of cities. The combination of deep learning and edge computing enables the cities to become more economic, green and efficient [54,55].

Wang et al. [56] proposed a method combining LSTM and CNN that can recognize people's different gestures. Therefore, users could control the events like turning on the light remotely. A deep learning mechanism in literature [57] facilitated people to detect the attack behavior and invalid data injection to the grid in real-time. To some degree, the smart grid is a scenario where edge computing and deep learning will shine since the distributed structure and massive data produced every second.

Smart Industry The two principles of smart industry are production automation and smart analysis. In industrial manufacture, the huge amount of data produced by tremendous sensors and devices could be processed well if edge computing is made the most of to make the industry more smart [58,59].

A system in paper [60] was used for machine health monitoring based on the combination of CNN and bi-LSTM. As a result, all machines can achieve the longest life from a global perspective. Wang et al. [61] proposed an architecture based on DNN that can predict the remaining energy as well as the lifetime of batteries.

2.3 Collaboration of EC and DNNs

In edge intelligence, EC and DNNs collaborate tightly and benefit from each other, which enhances the controlling of edge networks and improves the efficiency of task processing. Based on the relationship between EC and DNNs, the collaboration is reflected in two aspects: DNNs for EC and DNNs on EC [62]. For one thing, equipped with intelligent methods with DNNs, devices in EC could be organized or managed more intelligently, thus unleashing the significant potential and gaining great scalability of EC. And for another, by deploying DNN models in edge devices distributedly, the applicability of deep learning models is improved and DNNs could be processed more efficiently compared with the conventional distributed deep learning paradigm. Additionally, it is worthwhile explaining the relationship between distributed deep learning and edge intelligence. Distributed deep learning is a general processing architecture enabling computation to be executed on multiple devices. However, the communication cost, data security and resource allocation make it hard to determine an optimal policy efficiently in a naive distributed architecture. Hence, edge intelligence, based on distributed deep learning architecture, is proposed to solve the above challenges intelligently with the application of EC, thus achieving the high utilization of resources in edge networks. In summary, distributed deep learning is the fundamental base of edge intelligence and edge intelligence enables distributed deep learning to maximize its potential. Federated learning is another deep learning architecture with distribution, which can be viewed as an implementing technology to achieve edge intelligence in the co-training and co-inference in edge networks for efficient communications and data security of edge nodes. The further introduction of federated learning is represented in the following part.

2.3.1 DNNs for EC

As a distributed paradigm equipped with software-defined networks, EC can organize devices and provide services robustly and elastically. However, the heterogeneity and dynamicity of the EC scenario lead to the difficulty of the resource allocation and offloading decisions. To tackle these problems, deep learning strategies based on DNNs are proposed to obtain the optimal policy. Elgendy et al. [63] proposed an optimization scheme with deep learning to optimize the resource allocation problem with non-deterministic polynomial complexity. Considering that the states of edge devices and edge servers are various and the offloading modes of vehicles are different, the determination of an efficient offloading policy has become a challenge. Leveraging the deep Q-learning scheme, Zhang et al. [64] proposed an optimal offloading strategy to improve efficiency and reliability, which considered the determination of target servers and transmission modes. Equipped with DNNs, traditional reinforcement learning evolves into deep reinforcement learning (DRL) and is widely applied in the above resource allocation problems. To achieve a higher quality of service in a greatly dynamic environment composed of multiple agents, He et al. [65] proposed a resource allocation strategy based on multi-objective DRL considering multi-dimensional resources. In summary, deep learning-based strategies are leveraged to optimize the allocation of various resources and determine optimal offloading policy since they can extract deep features from practical scenarios. Since EC is a scenario consisting of massive devices and factors required to be considered are tremendous, deep learning strategies tend to perform better than traditional optimal algorithms of the powerful ability to extract features and fit functions. However, there are still some challenges in the processing of DNN-assisted optimization. Firstly, during the construction of models, models shall be formulated with limits and restrictions mathematically. Otherwise, the curse of dimension may lead to the failure of processing. Commonly, denoting constraints as penalties in the formulas of models and then combining them into system-wide optimization is a solution. Moreover, given the large volume of DNNs, they require a long delay to achieve the inference tasks. Hence, how to deal with the trade-off between optimality and efficiency shall be considered in depth according to different scenarios and requests [62]. Since how DNNs serve edge intelligence is not the focus of this paper, more information could be obtained in papers [10,33,62,66].

2.3.2 DNNs on EC

With the development of IoT and the surge of mobile communications, there are massive data are generated on the network edge. To serve better intelligent applications, more and more DNNs are deployed in the edge devices and exchange data frequently, which increases the pressure on the resources of devices [67,68]. Thus, DNN tasks requiring large computation or low delay can be offloaded to edge devices in a collaborative manner instead of processing them in a single resource-constrained device. Furthermore, as to DNN tasks processed with frequent communications in could servers, they shall be migrated to the edge. Hence, EC is applied to manage these DNN models deployed in edge devices for better utilization of resources and quality of service. In the aspects of DNNs on EC, the collaboration lies in that EC provides decentralized and distributed platforms based on edge devices, which enable the training and inference of DNNs to be processed efficiently [69]. To introduce this kind of collaboration in-depth, efforts on how DNNs achieve co-inference and co-training under EC will be presented in detail. The general ideas for combining complicated DNN models with EC are introduced as follows.

Co-Inference To provide mobile intelligent applications, some small DNN models are directly deployed in edge devices with model compression techniques and the inference tasks are completed locally by themselves. However, there are great challenges when large-scale DNN models are deployed in single devices since tremendous computation power and memory are required. Hence, empowered by EC, it is a rational and good choice to partition large-scale DNN models into several sub-models and send them to different edge devices, thus being inferred distributedly with collaboration. To achieve the above effects, two main techniques are applied: task offloading [70,71] and DNN model partitioning [31,72].

If the inference requests require a highly low delay while the computation power of the single device is constrained, the device can migrate computation partially or entirely to other devices. Therefore, the inference speed is improved and the energy consumption is saved as well, especially for resource-constrained devices. However, the determination of offloading policy shall consider both the acceleration payoff and transmission cost. Ran et al. proposed a framework for mobile deep learning called DeepDecision [73], which considered processing accuracy, energy constraints, network conditions and so on to achieve a system-wide optimization. This framework could adapt to variable network conditions automatically. Han et al. [74] proposed MCDNN, an offloading decision framework to optimize stream processing, which is based on approximate. This paper worked out how to serve heterogeneous request streams with restricted resources in an optimal manner. However, this work did not support the execution under model partitioning. As a real-time recognition system based on EC, Glimpse [75] could offload the computation of recognition algorithms to other devices with high computing capability for faster inference. FemtoClouds [76] could leverage other resource-constrained edge devices to accelerate the computation instead of offloading tasks to remote cloud servers, which saved massive energy and time for transmission. Benefiting from EC, resource-constrained mobile devices can be fully utilized, thus improving the scalability and reliability of the co-inference system without depending on single powerful servers. However, the great dynamicity, instability and heterogeneity will make it difficult for decision algorithms to converge and lead to some potential security challenges.

According to the structure of them, DNN models could be partitioned into several parts or blocks with several layers and the inference of each sub-model could be executed on distributed devices in EC. Although there exists the communication delay generated by intermediate results, the system-wide revenue of processing delay and energy consumption still makes it worthwhile to apply these strategies. As the extension of traditional offloading, before deciding how to offload the computation of sub-models, it is of great significance to determine how to partition the raw DNN models. Commonly, to adapt to the dynamic and heterogeneous environment in EC, device resources, network conditions, service requests, energy consumption and other factors are taken into consideration for optimal partitioning policy. Moreover, there are two mainstream partitioning methods: partitioning horizontally and vertically. CNN models can be partitioned vertically while traditional DNNs are partitioned horizontally, where several layers are executed on a device and other layers are inferred on other edge nodes. Neurosurgeon [31] is a framework that can partition DNN models in the grain of layers intelligently and automatically for lower delay and energy consumption. ECRT [77], a real-time object tracking system, partitioned a CNN into two parts and they were executed in the local device and edge device respectively to achieve the minimum power consumption of devices, considering the dynamic environment and delay requests. By partitioning large-scale DNN models into smaller sub-models, the capabilities of edge devices and resources in the edge networks could be considered and allocated more precisely and efficiently, thus easing some problems such as delay, energy consumption and overload. However, how to partition a model more reasonably and how to adapt to the dynamic EC environment better are still challenges.

In conclusion, EC enables the DNN inference more intelligently and efficiently by leveraging the resources of the EC system more optimally.

Co-Training In the conventional training mode like centralized cloud training, the data transmission cost and privacy problems are challenges of co-training. However, in decentralized EC, these issues could be solved by partitioning raw DNN models into several sub-models and they are trained on an edge node respectively and directly with local data. In this manner, the network burden resulting from data transmission is eased and private data security is enhanced since each edge node trains its sub-models. The partitioning strategy of raw DNN models is similar to the mentioned above, thus not repeating here. DNN partitioning, federated learning and transfer learning are three common and powerful techniques enabling co-training in EC and they will be introduced as follows.

Same as the DNN partitioning scheme applied in co-inference, DNN models are usually partitioned into several parts and the former part delivers partially processed data to the next sub-model. Matsubara et al. [78] proposed to distill the architecture of head sub-models after partitioning DNNs into head and tail parts, which were deployed in a local device and edge node, respectively. Hence, the intermediate data delivered to the tail part was simplified, thus easing the transmission load. However, in this manner, the process of distillation would impact the accuracy and transmission quality. Moreover, the determination of the proper splitting point according to the dynamic and heterogeneous environment still requires research efforts.

To ensure the data security and improve communication efficiency among edge nodes, federated learning [79] is applied in c-training with the server-client architecture, where the server averages models uploaded from clients. Since each client or edge node trains the local model based on the global model with local data, information security is ensured. In this perspective, federated learning can be viewed as a platform provided by EC and such a platform promotes the co-training, which reflects the collaboration of EC and DNNs as well. Wang et al. [80] proposed an adaptive federated learning strategy to optimize the trade-off between updating data from local devices and aggregating global parameters in the resource-constrained EC.

Transfer learning enables DNNs to initialize some parameters with weights learned from models pre-trained before and copes with problems that how to process distributed data. Zhang et al. [81] proposed to apply transfer learning to improve the service performance of edge devices with poor computing capacity and obtained a high improvement in system-wide efficiency. Cartel [82], a collaborative transfer learning system applied in an edge-cloud environment, aimed to facilitate edge devices with better adaptability to the situation changes. Compared with isolated and centralized training, the training time and transferred data decrease significantly.

3 Lightweight Models Applied in EC

Due to their limited computing and storage resources, many edge devices need to deploy relevant lightweight models or environments to implement distributed inference. In distributed inference, lightweight models could reduce the communication overhead with servers efficiently. Moreover, they are composed of fewer parameters so that devices download fewer data from the cloud compared with common models. For some embedded devices as well as mobile devices, lightweight models are easier to deploy. This section will introduce lightweight models in the view of realization strategies and application scenarios.

3.1 Strategies to Create Lightweight Models

To make heavy and large DNN models more lightweight for faster training and inference [83], there are four mainstream strategies: model pruning, quantization, low-rank factorization and knowledge distillation. Each of them will be introduced briefly below.

Pruning For the sake of improving the accuracy, more and more modern DNN models have become very large or with tremendous numbers of layers. However, the massive parameters always lead to an unbearable long time and huge storage, which makes it difficult to apply them in mobile environments, especially in the scenario of end-edge collaboration. DNNs usually include lots of redundant weights that could be removed without reducing the accuracy of prediction. Model pruning [84] is the method that simplifies heavy models to lightweight models able to be deployed in the end or edge devices with low computing power. Fig. 4 shows the basic implementation of pruning.

A DNN model could be pruned during the training or after it. Various techniques have existed such as weight pruning, neuron pruning, filter pruning, layer pruning and so on. In weight pruning, if the value of weight is lower than the threshold, the weight will be pruned or thrown away. Obviously, it makes sense since the low weight means a low contribution to the result. Neuron pruning saves a large amount of time by virtue of removing the whole redundant individual neuron instead of exact weights, which is time-consuming. After a neuron pruning, all connections with it will all disappear. As a type of DNN, CNNs also could benefit from model pruning. In filter pruning, while the 'importance' of each filter is calculated, they are going to be ranked according to it. Similarly to weight pruning, the least important filter is expected to be removed for its low contribution and high consumption.

Layer pruning is suitable for a very deep DNN in which even some layers could be seen as useless or redundant.

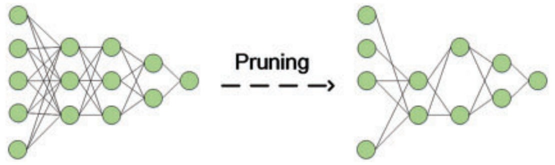

Figure 4: The schematic diagram of pruning a DNN

Quantization Normally, the weights of DNNs are presented as 32-bit floating-point numbers. Nevertheless, this representation leads to pressure on computation as well as storage. Reducing the number of bits can result in a huge reduction in the number of operations required, let alone the size of DNNs. Specifically, quantization [85] also reduces the transferring overhead in the scenario of edge computing collaboration. Inspired by the idea of weight quantization, we can also reduce the bits of other parameters involved in training or inference including the gradient and activation. Besides, there is another method to achieve quantization where weights belong to different clusters and they share the same value in the same cluster. And then, the processing of actual 32-bit weights could be simplified to fine-tune the shared weights. Fig. 5 illustrates the effect of quantization, which is a simple example.

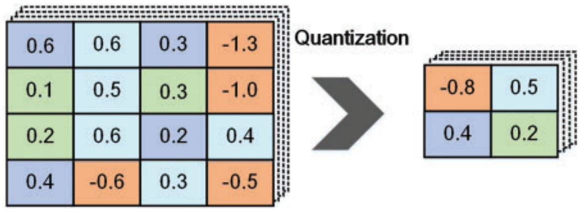

Figure 5: The schematic diagram of compressing a DNN by quantization

The strategy of quantization can be applied during the training or after it. During the training, the time consumption of learning could be saved a lot due to the fewer weights used for training actually. The efficiency of inference and the energy consumed both get improved significantly since the heavy DNN model is simplified when it is quantized after the training.

Low-Rank Factorization In this method, low-rank factorization [86], a weight matrix is replaced by several smaller dimension matrixes. As to feed-forward neural networks and CNNs, the popular technique to realize low-rank factorization and reduce the number of parameters is singular value decomposition (SVD). For any matrices A $\in$ $\mathbb{R}^{m\times n}$ whose dimension i $m \times n$, there is always a factorization: $A = USV^T$. Assuming that r is the rank of a matrice, $U = \mathbb{R}^{m\times r}$, $S = \mathbb{S}^{r\times r}$, and $V^T = \mathbb{R}^{r\times n}$. In the equalities above, S is a diagonal matrix in which the singular values are on the diagonal and U as well as V are two orthogonal matrixes. In this way, a large matrix could be factorized into smaller ones so that the storage requirement is improved. The process could be seen in Fig. 6. Meanwhile, considering that filters in CNNs are made up of matrixes, this technique is able to accelerate the processing. Besides, in fully-connected layers, low-rank factorization could be applied to reduce storage and accelerate inference. Given the frequent communication of devices in the end and edge, the cost of transferring weights and storing weights is improved by this strategy. In this view, heavy DNN models become more lightweight so that the performance of edge intelligence gets better.

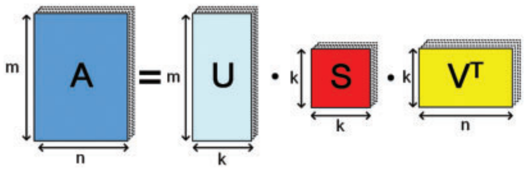

Figure 6: SVD–a scheme of low-rank factorization

Knowledge Distillation In knowledge distillation (KD) [87], the much heavier models, or teacher models, are trained on a very large dataset and then we transfer the knowledge learned by them to smaller models, or student models. That is to say, the final objective of the above processing is to equip student models with the generalization capability learned from teacher models while being more lightweight. Due to this strategy, some computation-intensive models which are not suitable for the resource-constrained devices in edge computing could be replaced with the smaller models with smaller sizes and computation without much loss of accuracy. It is worth noting that KD is different from transfer learning. The former is commonly used to realize model composition and get a lightweight model while in the latter we use the same model architecture and weights, only replacing some fully-connected layers. The diagram could be seen in Fig. 7.

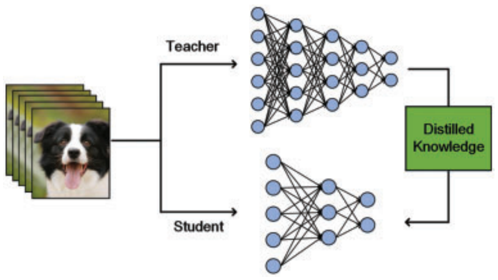

Figure 7: General process of knowledge distillation

The idea of KD matches the hierarchy architecture of edge computing. As to some quite heavy models, the cloud or powerful edge devices could pre-train the teacher models and then the knowledge could be transferred to end devices conveniently. Finally, the end devices or low-ability edge devices could apply the generalized small models to do some inference [88].

3.2 Application of Lightweight Models

There are several deployment scenarios for lightweight models in EC: image processing, privacy-preserving, prediction and monitoring. These models and their highlights are listed in Table 1.

Image Processing Nowadays, a huge amount of information is presented in the form of images and videos. However, some processing of images, such as recognition, classification, and picture enhancement, often require a large number of resources. But this is indeed a very luxurious behavior in the environment of edge computing. For example, in traffic, real-time monitoring and behavior determination requires a lot of analytical work. The deployment of lightweight models in surveillance devices to replace the original recognition system can effectively reduce the consumption of resources.

Image detection is a common kind of task in edge computing. Agarwal et al. [89] proposed a lightweight deep learning model for human activity detection on edge devices so that it could reduce the latency and cost of communication. The model was developed based on the shallow recurrent neural network combined with LSTM. In the future, this model could be extended to realize other more complex tasks and be deployed on more kinds of edge devices. A hybrid multi-dimensional spatial and temporal segmentation model for detecting defects was proposed by Hu et al. [90]. It had two core techniques: sequence-PCA layer as well as spatio-temporal and channel attention block. Compared with models for segmentation, this model yielded better performance and has much fewer parameters. In edge computing, many object detection models are sensitive to computing complexity and they need to be modified before being deployed commonly. Yao et al. [91] introduced CrossNet, a compute-friendly anchor-free detector. To improve the performance, it adopted a new sample matching strategy based on neighbor points. Although it is more than ten times smaller than the CenterNet, the performance is only a little weaker. In order to reduce the cost of computation, it is of great significance to generate redundant features. Zhang et al. [92] proposed the CSL module using a lightweight convolutional method to generate redundant features which could cost fewer FLOPs to approximate convolution-3$\times$3's fitting ability. Moreover, they built CSL-YOLO, the object detector with two lightweight components: CSL-Bone and CSL-FPN.

Recently, many image classification techniques have been combined with edge devices. Convolutional neural networks are of great significance in constructing a good classifier. Nonetheless, the large numbers of layers for accuracy make it difficult for the classifier to be realized on edge devices. To tackle this problem, Sharma et al. [93] proposed a lightweight image classifier–LightNet, which could extract more features from the input with fewer parameters by making good use of different receptive fields. It replaced the stand convolution with point-wise and depth-wise convolution. These two lightweight convolutional layers make up the principle blocks of LightNet–main block as well as the transition block. The former takes the input from the previous layer and passes it to the transition block through two paths. The latter based on DenseNet is responsible for the down-sampling. Experiments show that LightNet could perform better with fewer parameters on the CIFAR-10 dataset. Consequently, it is suitable to be implemented on edge devices.

Image classification usually needs a large number of parameters. However, deep learning models are prone to overfitting when tuning a large number of parameters with a limited number of labeled samples. Jia et al. [94] proposed a lightweight CNN (LWCNN) that could deal with a small sample set problems well for hyperspectral image classification. In this framework, a dual-scale convolutional (DSC) module was designed to represent data from different aspects, then a bi-channel fusion (BCF) module filtered the feature vectors from DSC layers. In computer-aided diagnosis, many deep learning-based methods are difficult to be implemented on edge devices due to their massive number of parameters and high computational costs. So, it is important to find a low-cost solution where models are compressed while maintaining accuracy. Kumar et al. [95] proposed MobiHisNet, a lightweight CNN model based on MObileNet, which was applied to histopathological image classification (HIC). It reduced the computational parameters efficiently by applying a range of depth-wise separable convolutions. Thus, MobiHisNet could extract features from histopathological images to contribute to the diagnosis. Compared with the state-of-the-art models, it is faster in image classifying. Additionally, experiments illustrated that it had better adaptation on edge devices in terms of accuracy, inference time, model size, and memory footprint.

There are also some models for improving the quality of images. Tang et al. [96] proposed a model for single image haze removal. It was an end-to-end system learning the mapping between the original images and the latent images without haze directly. Zhang et al. [97] proposed STAR, a lightweight structure-aware transformer, for enhancing images in real-time by capturing the long-range dependency of different image patches. For example, STAR could be used for photo retouching as well as illusion enhancement, which is indispensable for mobile phones.

In edge computing, since some sensors can not provide high-definition images and the quality of images is different, these lightweight models could accelerate the servers' processing by dehazing or enhancing the images locally for they are lightweight and can be deployed on edge devices. Moreover, due to these lightweight models or components, the whole edge system could realize distributed processing of images collaboratively.

Privacy Preserving With the prevalence of IoT and edge computing today, the massive amount of sensors and applications that access, collect, and compute sensitive user information provides convenience but also poses a threat to the privacy and security of users. Especially, when distributed inference is performed in edge computing systems, much private information will be exposed to other devices or servers for collaboration, which creates opportunities for malicious attackers. Combined with various recent studies, deploying suitable lightweight privacy-preserving models on edge devices is a very effective way to address the above problem.

Li et al. [98] proposed a lightweight data aggregation for privacy preservation, which could be deployed on edge devices with a constrained resource to protect the users' information. Different from the traditional schemes for data aggregation, this scheme could aggregate multiple target groups' data at the same time and update the aggregation rule list dynamically so that it is capable of protecting privacy practically and efficiently. In the scenario of diagnosis adopting edge computing, the diagnosis models requiring massive data will inevitably leak patients' privacy. To ease the problem, Ma et al. [99] proposed LPME, a lightweight diagnosis mechanism for privacy preservation. Instead of encrypting local data, this mechanism encrypts model parameters to remove the computation from ciphertext to plaintext. Therefore, it achieves lightweight privacy protection on edge devices when distributed inference or collaborative processing takes place.

E-health is another scenario where users' privacy is easy to be compromised. Zhang et al. [100] designed a data access scheme to address the security problems of e-health supported by edge computing, which adopts a lightweight encryption based on attribution. The algorithm outsources the data to be shared to the edge servers which are not trusted and then verified the correctness of results sent back while realizing the lightweight computation. The Mask Algorithm proposed by them cooperates with blind pairs to hide the bases and exponents of the data, thus achieving the preservation of privacy when outsourcing computation. In addition to medical aspects, various monitoring devices also have access to users' privacy. For example, home surveillance or public surveillance has been carried out in real-time monitoring, and if a malicious person intrudes, then a large amount of private information will be leaked. Fitwi et al. [101] proposed EnPec, a lightweight scheme for privacy protection, which could be applied on edge cameras robustly and securely. One component of EnPec is a lightweight frame classifier aiming to label frames as harmful or harmless according to the content of frames.

Monitoring and Prediction Intelligent monitoring and prediction as well as estimation have been playing a significant role in terms of production and life. However, since the sensors are so massive and they almost always transmit real-time images or videos, the transmission, storage and computation give a huge burden to the servers because of the edge devices' constrained source and power. Therefore, to ease the burden of edge servers even center servers, increasing efforts are made to design lightweight models or schemes able to be deployed on edge devices. Additionally, edge devices themselves also deserve to be monitored, such as rest-life prediction and production estimation.

Chang et al. [102] proposed a training framework to obtain prediction models for predicting the power output of solar energy. Compared with the traditional scheme where training is carried out on a single machine, this lightweight framework is more energy-efficient and suitable for devices with constrained power and resources. Due to the development of edge computing, the industry is transforming. Ren et al. [103] designed an edge-based method–lightweight temporal convolutional networks (LTCNs) that could predict the remaining useful life of devices in IoT, which is driven by data. In this method, the real-time prediction was obtained in the edge environment and then the prediction with higher accuracy could be obtained through the cloud. As a result, the method could reduce the processing time while the accuracy was not influenced. Deep learning technologies combined with edge computing also play an important role in public safety. Wang et al. [104] proposed a lightweight model based on the residual bottleneck block and dilated convolutional for crowd density estimation so that once accidents take place, feasible and efficient evacuation strategies could be developed in time. Compared with the state-of-the-art models, this model could compress nearly half of the parameters without much loss of accuracy. Zhao et al. [105] designed LCANet, an aftershocks monitoring system. To meet the requirements of real-time monitoring, it uses a lightweight context-aware attention network to detect the earthquake signal and pick phases. The computation requirements are reduced for the sake of it could be deployed to the edge devices. Moreover, this system is robust and could be generalized to other platforms easily.

Table 1: Several lightweight models and their highlights as well as effectiveness

Models	Highlights	Effectiveness
Agarwal et al. [89]	Reduce the latency and communication cost with great extension.	Accuracy: 95.78%
Hu et al. [90]	Have good performance in defect detection with fewer parameters.	Average F-score: 89.67%
Yao et al. [91]	More lightweight than CenterNet but has slightly less performance.	Model compression: 12.8x Achieved mAP: 74.2%
Zhang et al. [92]	Perform better with fewer FLOPs and parameters since it is compute-friendly.	Saved FLOPs: 43% Compressed parameters: 52%
Sharma et al. [93]	Make good use of different receptive fields with point-wise and depth-wise convolution.	Parameter size: Only 26 MB Classification accuracy: 90%
Jia et al. [94]	Good at dealing with small sample set problems with robustness.	Average accuracy: 93.77%
Kumar et al. [95]	Realize histopathological image classification efficiently and be lightweight enough to be applied on edge devices.	Accuracy: 83.79%∼90.19%
Tang et al. [96]	Achieve the haze removal of single image in end-to-end systems by learning the mapping between it with the originals.	PSNR: Higher than average SSIM: Nearly average Model size: Only 707.00 KB
Zhang et al. [97]	Enhance the images in real-time by capturing their dependency of different image patches.	Compressed parameters: 75% Improved PSNR: 1.8 dB

(Continued)

Table 1 (continued)

Models	Highlights	Effectiveness
Li et al. [98]	Aggregate multiple target groups' data concurrently and update the rule list dynamically to protect privacy.	Saved computation time: More than 50% Saved communication overhead: More than 60% even 80%
Zhang et al. [100]	Adopt a lightweight encryption based on attribution.	Encryption efficiency: 1.5x
Fitwi et al. [101]	Provide a framework to classify the harmful or harmless frames.	Average accuracy of capturing harmless frames: 96.04%
Chang et al. [102]	More energy-efficient and fit edge devices with constrained power and resources well.	Acceleration: Nearly 2x
Ren et al. [103]	Reduce the time for computation while not reducing the accuracy.	RMSE: Nearly 0.035%
Wang et al. [104]	Improve operational efficiency considerably with the same-level accuracy.	Compressed parameters: 50% Accuracy loss: Less than 10%
Zhao et al. [105]	Suitable for edge devices with low computing power and have good portability.	Model size: 3.7 MB

4 Key Techniques to Achieve Distribution

Deep learning has been playing a significant role in large numbers of fields such as medicine, finance, security, and so on. With its penetration, the data quantity of applications is undergoing a tremendous increase and the trend is still growing. Meanwhile, the complexity of computation and models' parameters are booming, thus leading to the higher consumption of training and inference time. Considering recent research [106,107], adopting distributed deep learning training and inference is one of the feasible solutions to deal with this problem.

Distributed deep learning works mainly in the form of parallelism, in which several devices work for the same target while doing different parts at the same time, such as synchronously or asynchronously. In parallel processing, workers need to synchronize every gradient to an exact gradient, then they could generate and process the next gradient [106]. However, the highly frequent synchronizations of gradient limit the scale of algorithms based on gradient descent. Data parallelism and model parallelism are two main schemes for distributing computation to different devices in the distributed system, especially in edge computing. In the scenario of edge intelligence, many devices collaborate to achieve the goal task by doing each one's assigned sub-task. The processing of achieving sub-tasks together is the processing of distributed deep learning.This section will first introduce the hardware foundation of distributed or parallel processing and then present the implements as well as some applications of the two parallel schemes.

4.1 Hardware Foundation

A CPU (Central Processing Unit) [108] is the core of a computer, which is composed of arithmetic and logical units (ALU), caches, and control units. The common structure of a CPU is shown in Fig. 8. CPUs are involved in general-purpose computing where computation is usually not massive but complex. So, CPUs require adequate controlling units to realize complex data controlling and transfer data while operating and caches to storage some results which are just worked out or will be used soon later. Due to the feature of their architecture, CPUs are suitable for scenarios where complex tasks like moving to control and managing the sequence between instructions take place.

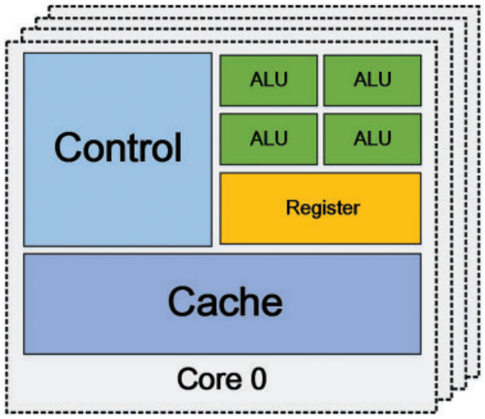

Figure 8: The common structure of a CPU

A GPU (Graphic Process Unit) consists of a set of multiprocessors, each of them having its stream processors and sharing the memory [109]. The stream processors can execute integer as well as single-precision floating-point operations. Additionally, they also own cores for double-precision operations. Since all multiprocessors can access the global device memory and they execute thousands of threads at the same time, the latency produced by accessing memory could be ignored. Register and resources sharing memory are partitioned by the threads being executed. The multi-stream architecture enables GPUs to do parallel computation where every step is independent of others. Fig. 9 shows the diagram of a GPU.

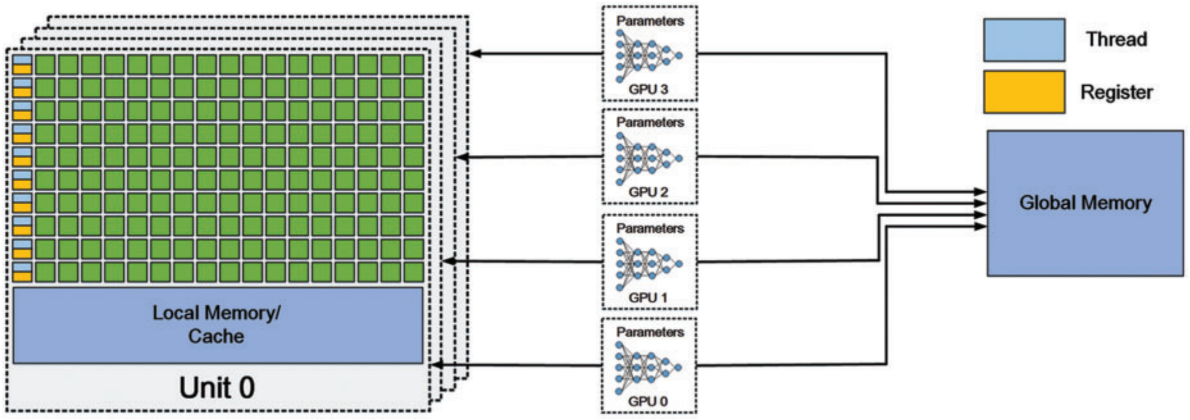

Figure 9: The common structure of a GPU

Compared with CPUs, GPUs have fewer components for controlling and caching while most of them are ALUs. As a result, CPUs do better in controlling and doing complex arithmetic while GPUs are more suitable for achieving the simple and repetitive. This feature makes GPUs widely applied in processing parallel arithmetic, which is the core operation of training deep neural networks or doing deep learning inference.

Constructing CPUs clusters has also been proposed to train deep neural network models. By leveraging the powerful computing power of clusters and the features that models can be stored distributively and parameters can communicate asynchronously, this scheme presents huge potential to compete with GPUs. Once a deep learning model is too large to be contained by a GPU, using the CPUs cluster teed to be a better plan. With 1000 CPU servers, the scheme in Dean et al. [30] achieved the training of deep neural networks by using model parallelism and data parallelism. Inspired by the advantages of GPUs and CPUs clusters, it seems a feasible and efficient solution that constructs GPUs clusters on the base of the CPUs-GPUs system. In this scheme, the large scaled training can be accelerated by playing to the strengths of single nodes' high performance as well as multiple servers' collaboration.

4.2 Data Parallelism

Before introducing the principle of data parallelism, the definition of parameter server needs to be learned about [110], whose diagram is shown in Fig. 10. Severs and workers are the two components of this architecture. Servers maintain the whole or partition of parameters shared globally, and synchronize the weights of every worker node. The number of server nodes is usually the same as the hosts' and each of them will operate as part of the synchronization. Worker nodes play the role of realizing the primary computation of algorithms, namely, some simple and low-level arithmetic like back propagation, convolutional computation, weights updating, and so on. They update and receive gradient only from server nodes and do not communicate with each other. Thanks to the parameter server, some deep learning algorithms can be accelerated nearly linearly. Additionally, the parameter server is a definition in logic, which may be not an independent server.

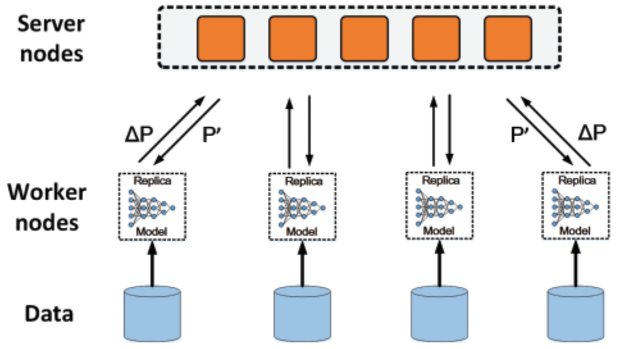

Figure 10: The architecture of a parameter server system

In data parallelism, data are distributed or partitioned into different workers, where complete models are deployed. That is to say, a worker node runs a whole model but processes data differently from others. Data parallelism achieves data exchange by the parameter servers mentioned above. The diagram of data parallelism is shown in Fig. 11. During the whole process of training and inference, every sub-process of each step is independent since servers communicate with worker nodes, respectively. In edge computing, many scenarios need and suit data parallelism since that many devices cannot carry these huge amounts of data. For example, traffic monitoring equipment is always deployed with the whole model while they could collaborate on a task due to the massive data that they cannot afford.

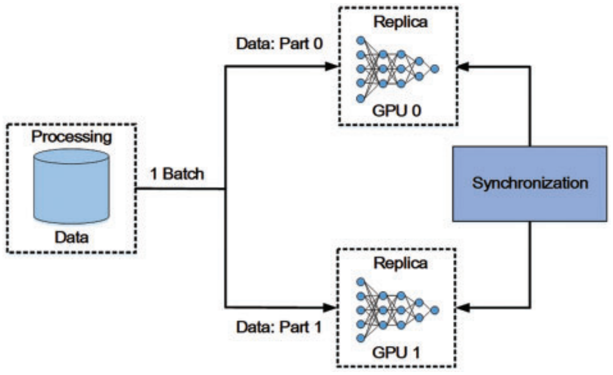

Figure 11: The diagram of data parallelism

There are two modes of data parallelism–synchronous and asynchronous modes. In synchronous mode, all the procedures train the data of the same batch and then exchange parameters when they all complete the computation. After exchanging parameters, procedures can start the training of the next batch on the base of the same new model. However, in asynchronous mode, if a procedure completes the training of a batch, it will exchange the parameters with parameter servers immediately not caring about other procedures. So, the latest result of a procedure will not be reflected in other procedures until the next exchange. In practice, since GPUs are connected with PCIe links, the synchronous mode is preferred.

In the rest of this subsection, some frameworks, models, or libraries will be presented for relevant workers to do further applications or research.

NOVA NOVA [111] is a functional language and compiler that is significantly useful in multi-core CPUs and GPUs. It is a polymorphic and statically functional language with higher-order functions used to express parallelism which includes scan, map, and reduce. The NOVA compiler is an optimizing compiler, lightweight but powerful. The NOVA compiler produces target codes from the NOVA language. It is stand-alone and can be embedded in other applications or used as a target for specific languages easily. Existing code could be integrated into NOVA programs thanking foreign functions. Nonetheless, foreign functions are considered to be of no side effects.

Over the last few decades, a large number of programming systems have been invented to make it more accessible for parallel programming to be used on multi-core CPUs and GPUs, including CUDA [112], OpenCL [113] and TBB [114]. These systems are targeted toward programmers familiar with the theory of the underlying parallel hardware and proficient in fine-tuning applications to improve performance. However, the feature that they are close to the bottom of hardware also becomes an obstacle for others to use them. Compared to them, NOVA is a system with high-level abstraction and performance. Meanwhile, NOVA has almost the same performance.

In summary, NOVA has the following two characteristics: recursion and type inference. NOVA could realize recursion by using μ-expressions, which are like fixed-point combinators. The NOVA language operates Hindley-Milner type inference [115] and does some extensions to support the polymorphism of some built-in parallel operators.

MALT MALT [116], a library integrating with existing machine learning algorithms and software, enables machine learning applications to realize distributed learning or inference leveraging data parallelism. By providing abstractions for memory updates that are fine-grained, it reduces the cost of data movement when incremental models change or update. With its general API, the developers can provide distributed applications written in C++ and Lua with data-parallelism, which are based on support vector machine, matrix factorization as well as neural networks.

During the distributed inference, the model replicas operate training on different sets of cores across nodes in a parallel way. Machine learning libraries use the vector library of MALT to produce the parameters or gradients of models that require to be synchronized among machines. Moreover, it loads the data of model replicas from distributed file systems like NFS and HDFS. Developers, use the API to input, send, and receive data across model replicas so achieve the synchronization of algorithms.

In conclusion, the existence of MALT makes various machine learning software leverage data parallelism efficiently, and the framework itself reduces the cost of network processing and transmission overhead.

AsynGraph In recent years. iterative graph algorithms have been proposed to be operated by accelerators based on GPUs. Nonetheless, existing methods do not make good use of the parallelism of GPUs, which limits the process of iterative algorithms. AsynGraph [117] was proposed to maximize the data parallelism of GPUs to accelerate the convergence speed of the process. Firstly, it is an asynchronous system with highly efficient structure-aware ability. AsynGraph extracts a graph sketch consisting of the paths between vital vertices from the original graph and treats it as a bridge where states could propagate fast. In this way, the states of most vertices can be conducted parallelly on the GPUs. Therefore, the utilization of GPUs is improved since the important vertices are processed efficiently. Moreover, AsynGraph adopts a forward-backward intra-path way of processing, which could handle every path's vertices asynchronously to boost the propagation speed while reducing the cost of accessing data. The research about how to extend it to heterogeneous platforms for the process on a larger scale has been carried out, which means the further exploitation of the high parallelism of GPUs in graph processing.

In summary, data parallelism is penetrating various aspects of distributed learning or inference, from whose underlying implementation to its conduction on different devices.

4.3 Model Parallelism

Model parallelism is another way of parallelism. Different from data parallelism's operation on data, model parallelism partitions a large model into smaller model layers among GPUs [118], namely, every GPU in the system only needs to be responsible for the assigned model layers' weight. The diagram of model parallelism is shown in Fig. 12. One model layer's output is the input of the next model layer and the gradients produced by backpropagation are transferred from one sub-model to another. These intermediate data are transferred among GPUs. Unfortunately, due to the dependence among each sub-model, if model parallelism is implemented in a naive way where only one single GPU works at a time, the efficiency and utilization of GPUs are very low. To tackle this problem, PipeDream [119] realized the pipeline of model parallelism by injecting several mini-batches into the pipeline at the same time. As a result, every GPU can operate its training and inference simultaneously.

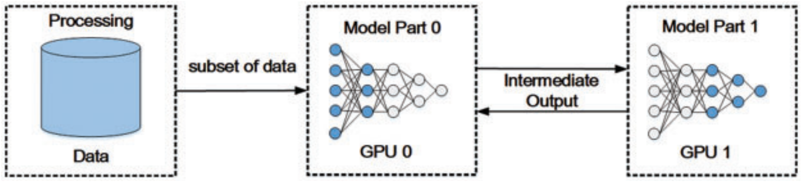

Figure 12: The diagram of model parallelism

When it comes to model parallelism, there are two main paradigms–layer-wise pipeline and tensor computation [120]. In pipeline one, the operations are performed on one sub-model or device before they are completed and outputs are transferred to the next device in which the further computation is performed. Harlap et al. [119] proposed an approach where a parameter server is combined with pipeline model parallelism. Nonetheless, these methods need to bear the suffering of inconsistency issues. Leveraging the synchronous gradient decent, Huang et al. [121] overcame the above issues. However, the additional logic for handling pipelining of communication and computation leads to reduced efficiency or changes to the optimizer itself.

Distributed tensor computation is a more general method, in which a tensor operation is partitioned to several devices to accelerate computation or to improve the scale of original models. FlexFlow [122] is a framework that provides a method to find the best strategy for parallelism. Shoeybi et al. [120] made the use of parallelism of transformer's attention to achieve the parallelism of their model.

Since accessible devices in edge computing are usually heterogeneous if they deploy the same model, their utilization of them is very low and each device's advantages may be wasted. For example, different devices can be assigned models of the appropriate size according to their capacity. On this basis, it can be further adjusted according to the actual situation. In this way, the overall benefits of edge computing can be further optimized.

SpecTrain Although data parallelism is the most common method for parallel computation of distributed inference, its high cost of communication between GPUs is still a serious issue. Pipelined model parallelism is an approach that reduces the communication cost but suffers from the weight staleness issue, namely, this approach leads to instability and the loss of accuracy since it uses stale weights. SpecTrain [118], a new weight prediction technique, is proposed to be applied in a pipelined method to solve the staleness and accuracy loss. The method efficiently improves the utilization of GPUs and maintains the same level's accuracy.

During the training procedure, the same version of a mini-batch will be used in a whole round trip, however, which makes GPUs have to queue for old versions so that wastes memory space. However, GPUs could adopt future weights instead of waiting for the earliest version of weights due to the prediction of SpecTrain. Specifically, if a mini-batch completes the round trip at time t, it could predict the t-version of weights in the early stage and perform computation with this version. The specific implementation of this approach is omitted here, and the details can be found in the papers cited in this section.

In edge computing, the massive edge devices will produce a long queue if they are processed in the traditional model parallel method and the waste, as well as waiting time, are catastrophic. So, SpecTrain is expected to be a powerful tool to ease the problems.

LAMP Large deep 3D ConvNets with Automated Model Parallelism (LAMP) [123] is proposed and it is feasible for large deep ConvNets to be trained using a large input block or even the whole image with LAMP. As is shown in the experiments, the accuracy of image segmentation could be improved by increasing the model size or input context.

It can reduce the inference time and improve the accuracy when the size of the input is very large, which seems to be contradictory. It benefits many areas where large image input is inevitable such as the analysis of medical images and 3D architecture search.

GEMS GEMS [124] is a memory-aware system of model parallelism based on GPUs. There are designs of GEMS: GEMS-MAST, GEMS-MASTER, and GEMS-Hybrid. Memory-Aware Synchronous Training (GEMS-MAST) makes use of the free memory and the computation is produced when GPUs have completed their forward and backward passes to train the same DNN's replicas in an inverted manner. Memory-Aware Synchronized Training with Enhanced Replications (GEMS-MASTER), a generalized version of GEMS-MAST, makes it possible for researchers to train a model with any batch size but on resources of the same numbers. GEMS-Hybrid, which combines data parallelism and memory-aware model parallelism, takes the advantage of data parallelism's near-linear speedup to accelerate the training.

This scalable hybrid system for out-of-core parallel computing contributes to various fields requiring large-scale training, promoting the popularity of distributed inference in edge computing.

4.4 Convergence of Data and Model Parallelism

Nowadays, for the sake of leveraging both parallel strategies at the same time and make the most of them, the convergence of them has been taken into research [30].

HetPipe HetPipe [125] is a DNN training system for large models' training on a heterogeneous GPU cluster which consists of low-level GPUs that could not achieve training independently. Its basic architecture is a heterogeneous pipeline, integrating data parallelism and pipelined model parallelism to take advantage of them. In the process of real training, a virtual worker composed of a group of GPUs processes mini-batches in a way like pipelined model parallelism, and multiple workers perform in a data-parallel way to improve the efficiency. To facilitate the convergence of parallelism of virtual workers, WSP, a novel model for parameter synchronization, is proposed as well. Experiments have proved that HetPipe is feasible and efficient and achieve 49% faster than the latest data parallelism technique.

In the real world, large numbers of edge devices have no powerful hardware for standalone training or inference and they are almost heterogeneous in an edge computing system. So, Hetpipe seems to be a good solution for utilizing these fragmentary GPUs to the most degree.

Model Dataflow Graph Pal et al. [126] proposed a method for hybrid parallelism. In this method, every worker with data parallelism consists of several devices and it partitions the model dataflow graph (DFG) with model parallelism, then allocates the partitioned to multiple devices. To the shortage of data parallelism, the increasing number of epochs resulting from the increasing size of the mini-batch will lead to a longer time of training and the acceleration could not be extended well when there are too many devices. So this method of hybrid parallelism tries to solve these problems by making every worker with a data parallelism model parallel as well in each device. During the running time, when the efficiency of data parallelism reaches a bottleneck, the hybrid parallelism will extend more devices to train.

To find the time when hybrid parallelism needs to be applied, researchers also develop a framework for analyzing the crossing point of devices. It is proved that this method performs than schemes with data parallelism only when training DNNs of different scales.

4.5 Comparison of Two Parallelism

In this subsection, data parallelism and model parallelism are compared in three aspects and their prospects of them will be introduced. The category is according to the paper [118]. The diagram of the collaboration of data and model parallelism is shown in Fig. 13.

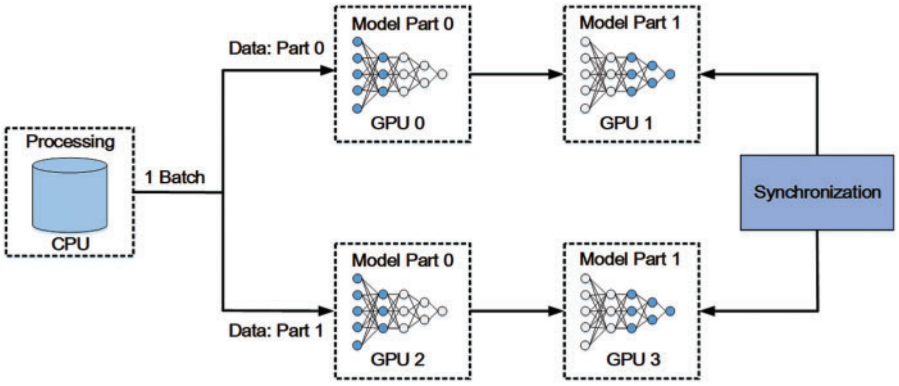

Figure 13: The convergence of data and model parallelism

Load Balance Across GPUs Since the implementation of data parallelism is to partition data into several sets across multiple GPUs, load balance is easy to maintain. However, it is very hard for model parallelism to allocate load in a balanced way because different DNN layers vary complexly. To solve this problem, Harlap et al. [119] proposed to use dynamic programming to achieve the balanced partition of models. Mirhoseini et al. [127] used reinforcement learning to partition models dynamically when they are running.

Inter-GPU Communication Although both of them require inter-GPU communications while running, model parallelism usually communicates much less than data parallelism because it has few intermediate parameters between layers. The frequent communication of data parallelism results in the slowdown with experiments showing nearly a quarter of the time is spent between inter-GPUs.

Training Efficiency For data parallelism, the mini-batch size decided by data partition influences the training efficiency and accuracy. Researches in [128,129] showed that if the mini-batch is large, the utilization of GPUs will increase but the accuracy will decrease. For model parallelism, the problem mainly lies in the staleness since the recent popular training is in the pipelined way where the latter mini-batches adopt stale weights to produce gradients, which decreases the steadiness and accuracy of training.

Considering the existing problems of data and model parallelism, future research is expected to be around the optimization of data partition and the staleness issues in pipelined model parallelism. Additionally, if approaches for improving the scalability of data parallelism are designed, data parallelism will show more significance in the booming scale of multiple-GPUs clusters. Moreover, the combination of data and model parallelism is the trend of distributed parallel computing, in which case the two methods can maximize their advantages.

5 Applications of Distributed Inference in Edge Intelligence

At present, many methods based on deep learning are proposed to optimize the processing of edge computing, such as the decision that how to allocate computation and storage to each device. For example, a computation offloading strategy is proposed in paper [130] by deep reinforcement learning;

augmented reality is processed by taking advantage of edge intelligence with reinforcement learning [131]; the energy consumed by computation is reduced by a game-based reinforcement learning and the efficient is improved [132]. In edge computing, massive numbers of edge devices could be applied as workers for the distributed training or inference of large DNN models, which are used to be processed only in cloud servers. In this way of combining distributed deep learning with edge intelligence, the whole system could gain lower latency, higher energy efficiency, and better utilization. The basic structure of distributed processing in edge computing is in Fig. 14 and there are many other realizations of this structure. In this structure, many heterogeneous devices are accessible in the edge scenario, so they are supposed to collaborate in the form of data parallelism or model parallelism and even hybrid parallelism according to the practical context of the scenario.

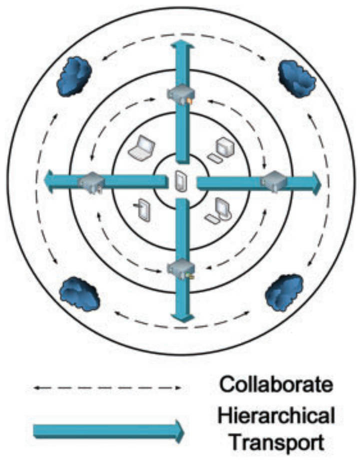

Figure 14: The structure of distributed processing in edge computing

Typically, under the control or allocation of edge intelligence, raw data (input data) may be distributed to edge devices and the target tasks are going to be processed. In this stage, the controller device needs to determine the way of parallelism and determine whether cloud servers are required to help process task according to the practical situation. To satisfy the requirements above, cloud servers need to deploy relevant models in advance. Many factors impact the decisions, such as the characteristics of tasks, the quantity of intermediate data, the quality of the network, etc.

DeeperThings When it comes to image classification, the inference of huge CNN models is harder to be executed on a single device, especially in edge computing and IoT where devices are recourse-constrained while tasks are heavy. However, although a single device is not powerful, we can make the most of the number of devices. That is why distributed inference makes sense. DeeperThings [133], the improved version of DeepThing [134], is proposed in this background. It is a method that supports the CNN inference's full distribution by partitioning layers that are fully connected and intensive in weights as well as features. Moreover, the team also achieves the joint optimization of computation, memory, and communication demands on the base of distribution. Moreover, a novel approach to integer linear programming (ILP) is designed. As is shown in experiments, with the integration of layers, this method could balance the memory footprint among devices and reduce the communication demand by over a quarter. According to experiments, the integration of layer fusion helps reduce the demands of communication by 28.8%, which results that the inference task being accelerated by 1.52x compared to naive layer partitioning. Although model partition has been a technique to achieve parallel processing, some novel methods could improve them and then have better effects.

To some degree, this scheme reveals the general idea of dealing with large CNN models in scenarios rich in computation- and recourse-constrained devices. Moreover, since partitioning layers is the core operation in it, this scheme may apply data or model or hybrid parallelism to achieve further improvement. For example, the devices with similar power and recourse could operate similar computations in parallel to improve the utilization of the whole system.

DeepHome The smart home is a promising scenario where edge intelligence plays a significant role while the tasks of inference on various devices are also a problem.

DeepHome [135], a novel distributed inference system, is capable of allocating the tasks of machine learning inference to heterogeneous devices distributively, aiming to improve the integral low latency and privacy preservation and achieve the most utilization of devices in the home. Considering the present and future situation of the intelligent home, this system is of great enlightenment.

The architecture of DeepHome is composed of two main modules: the resource negotiator and the scheduling module. The resource negotiator is responsible for managing devices' resources, tracking the working states of devices and maintaining the network connections of devices. The other module is used for scheduling the tasks of devices and making distribution plans, which consists of components for task admission, task monitoring and task scheduling. There is a selected centralized node as a 'registry' for the management of the whole system, whose work includes the dynamic distribution of tasks to suitable devices. For example, tasks will be allocated to free devices when others are occupied or operating their primary function and a complex task that consumes a huge amount of recourse and time will be distributed to multiple devices for accelerating.

The experiment shows that their scheduling algorithm makes the most of scheduling delays of less than 20 ms, which could satisfy daily requirements. Moreover, most of the inference delays lie between 200 and 400 ms, which is also acceptable. As a result, edge intelligence has the potential to collaborate with distributed processing well.

For further improvement, some specific lightweight models can be deployed to devices according to the personalized environment of the home. It can be foreseen that with the progress of hardware, parallel schemes will optimize the distributed tasks.

HierTrain Liu et al. [136] proposed HierTrain to accelerate the DNNs processing on the MECC (Modele-Edge-Cloud Computing), which overcomes the drawback of consuming a long time and requiring massive resources when transferring data from the edge to center. HierTrain, a hierarchical framework for edge AI, deploys the training tasks of DNN on the hierarchical architecture efficiently. The key to this framework is the hybrid parallelism strategy, which assigns the layers of DNN and data samples in an adaptive way across the edge devices, edge servers and the cloud center. To obtain the scheduling policy under the parallel strategy, they formulate the scheduling of computation as an optimization problem for minimization.

According to the experiment, HierTrain can achieve up to 6.9x speedups compared to the training approach based on cloud hierarchical.

Compared with other applications of parallel strategy, HierTrain assigns the data and model layers to three levels which consist of different types of devices. This novel scheme pioneers the more granular applications of parallelism considering the hierarchical architecture of edge computing.

Systems Considering Privacy and Security In edge computing, the massive communication between edge devices may also bring the problems of a privacy breach. One of the solutions is to apply the distributed deep learning system in edge computing, which could provide efficient and reliable protection for local users' private data.

Saputra et al. [107] proposed a novel framework, allowing model edge nodes (MENs) to cooperate and exchange their information to improve the accuracy of the prediction of content demand, to realize the highly accurate and secure prediction. In this framework, MENs are only allowed to exchange the gradients and privacy is preserved well. To some degree, this framework achieves a very fine-grained scheme for privacy preservation in edge computing by adopting distributed processing, which has far-reaching significance since the booming of edge devices is inevitable in edge computing. Moreover, for more personal protection, each device may construct its lightweight model locally for better performance detection of invasion.

Kozik et al. [137] proposed a scheme for attack detection based on distributed ELM (Extreme Learning Machine) leveraging the HPC clusters to deal with training requiring a long time and huge computation. In this scheme, the construction of large-capacity models takes place on the cloud with adequate resources so that classification models resulted could run on multiple devices in the whole system of edge computing.

Summarized from the above applications, we can see that distributed processing could offer further improvement to the scenario of edge computing by providing more fine-grained service.

6 Challenges and Prospects

6.1 Challenges

Since there is great potential in combining edge intelligence with distributed deep learning, further research needs to be done for addressing the current problems, such as the sensitive data of users, latency due to network congestion, offload failure due to insufficient computing and storage capacity, etc. To enable parallel strategies to cooperate with distributed processing well in edge computing, there are still several challenges to overcome.

• Scalability: The construction of the distributed strategy when devices are highly heterogeneous in a cluster still has space for improvement. Although there are large numbers of devices available in the scenario of edge computing, they are usually highly heterogeneous so it is difficult for the distributors to make proper plans of distribution with the consideration of each device's cost and utilization. If excellent strategies applicable to groups full of heterogeneous are not found, the scale of devices can contribute to the distributed processing concurrently to further improve the allowed scales of DNN models and data will be limited. A scheme proposed in [138] does an attempt in the aspect of model parallelism and adapts the distribution to heterogeneous devices well. Strategies based on other forms of parallelism such as hybrid parallelism need to be proposed to take the most advantage of parallelism and distribution.

• Fault Tolerance: Since universal failures are inevitable, the abortion and exit of tasks have become a problem. Therefore, it is vital to monitor the state of each device while the cost increases. Moreover, transferring the tasks done partly to other devices without influencing the whole efficiency intolerably is also an issue. However, when the number of devices or working nodes is passed a certain quantity, the probability of failures will reach a very high level and the continued abortion could damage the training or inference. As is proposed in paper [139], very specific hardware could be used to decrease the probability of failure while high expense and hard access make this scheme effect little. As a result, strategies or hardware that could deal with the problem of failure needs to be researched at present. Moreover, beyond failure detection, activities such as transferring tasks taken after which are also important.

• Privacy: The problems of privacy and security of each device in the system of distributed processing remain to be addressed especially in edge computing where devices are connected tightly. From now on, while there have been some theoretical schemes to ease the problems of privacy, current frameworks do not perform well in supporting even basic privacy [140]. In edge computing, the privacy of devices or users is the foundation of any applications so the research of frameworks or schemes considering privacy while distributing the target models is of great priority.

• Performance: The performance of distributed processing is not so good when the number of devices is significantly large. In the reality, it seems that fast training will always benefit if the resource used is sufficient. For example, the framework in paper [30], which utilizes thousands of machines to train large DNN models, achieves great acceleration in training. However, although the distributed use of GPUs brings high performance, the efficiency is usually below 75%. So, more efforts of research can be taken to explore the joint optimization of efficiency and performance.

6.2 Prospects

Since edge computing has been integrated into various aspects of life, for further improvement and development, distributed processing is expected to take the most advantage of edge computing in several fields. Now that the popular fields of edge computing have been introduced clearly in detail in the section about the background, the prospects of the combination of them will be presented briefly below:

• Smart Multimedia: Nowadays, data and computation coming from real-time multimedia have made some image or video processors deployed with large-scale models tired especially during peak periods. Therefore, the distributed processing of compression or decompression of images and videos is much needed. In this field, distributed processing can combine with edge computing well for the scenario is usually abundant in edge devices.

• Smart Transportation: Automated driving and the transmission of information on the road are two main directions for researchers. The computation produced by automated driving can be partitioned to other devices if cars have poor power or their resource is constrained. Additionally, passengers' applications also add a burden to the servers, whose DNN models are large and real-time input data are massive as well. As a result, many efforts need to be made in this direction.

• Smart City: A city can be viewed as a complex system that contains all kinds of devices. So, distributed processing should explore the diversity and maximize the utilization of free and available devices. For example, some blocks are of few activities at night, so the devices can be leveraged by the tasks that are still active.

• Smart Industry: In some large-scale factories where machines are connected to be integrated into edge computing, the monitoring and prediction of machines can be processed in a distributed way to achieve the balance of the whole system. Moreover, considering the feature that machines are usually the same in a group or cluster, distribution and parallelism are easier to achieve.

7 Conclusion

The convergence of distributed processing and edge computing significantly accelerates the training and inference of modern large-scale DNN models. This paper aims to provide an enlightening review of distributed processing schemes based on data and model parallelism in edge computing. The main background of DNN and edge computing is introduced in the beginning. After the description of lightweight models deployed on distributed devices, the research and applications of this scheme are presented. Eventually, several challenges and prospects of distributed systems in edge computing are discussed.

Funding Statement: This research is supported by the Natural Science Foundation of Jiangsu Province of China under Grant No. BK20211284, the Financial and Science Technology Plan Project of Xinjiang Production, Construction Corps under Grant No. 2020DB005 and the National Natural Science Foundation of China under Grant Nos. 61872219, 62002276 and 62177014.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Muhammad, K., Khan, S., Ser, J. D., Albuquerque, V. H. C. D. (2021). Deep learning for multigrade brain tumor classification in smart healthcare systems: A prospective survey. *IEEE Transactions on Neural Networks and Learning Systems, 32(2),* 507–522. DOI 10.1109/TNNLS.2020.2995800.
2. Xu, X., Tian, H., Zhang, X., Qi, L., He, Q. et al. (2022). Discov: Distributed COVID-19 detection on X-ray images with edge-cloud collaboration. *IEEE Transactions on Services Computing, 15(3),* 1206–1219. DOI 10.1109/TSC.2022.3142265.
3. Gulshan Kumar, H. A. (2022). Deep learning-based cancer detection-recent developments, trend and challenges. *Computer Modeling in Engineering & Sciences, 130(3),* 1271–1307. DOI 10.32604/cmes.2022.018418.
4. Sadr, H., Pedram, M. M., Teshnehlab, M. (2020). Multi-view deep network: A deep model based on learning features from heterogeneous neural networks for sentiment analysis. *IEEE Access, 8,* 86984–86997. DOI 10.1109/ACCESS.2020.2992063.
5. Adege, A. B., Yen, L., Lin, H. P., Yayeh, Y., Li, Y. R. et al. (2018). Applying deep neural network (DNN) for large-scale indoor localization using feed-forward neural network (FFNN) algorithm. *2018 IEEE International Conference on Applied System Invention (ICASI)*, Chiba, Japan. DOI 10.1109/ICASI.2018.8394387.
6. Chen, J., Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE, 107(8),* 1655–1674.
7. Li, H., Ota, K., Dong, M. (2018). Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network, 32(1),* 96–101.
8. Huang, Z., Liu, F., Tang, M., Qiu, J., Peng, Y. (2020). A distributed computing framework based on lightweight variance reduction method to accelerate machine learning training on blockchain. *China Communications, 17(9),* 77–89. DOI 10.23919/JCC.2020.09.007.
9. Zhang, M., Zhang, F., Lane, N. D., Shu, Y., Zeng, X. et al. (2020). Deep learning in the era of edge computing: Challenges and opportunities. *Fog Computing: Theory and Practice*, 67–78. DOI 10.1002/9781119551713.ch3.
10. Marchisio, A., Hanif, M. A., Khalid, F., Plastiras, G., Kyrkou, C. et al. (2019). Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Miami, FL, USA. DOI 10.1109/ISVLSI.2019.00105.
11. Xu, X., Jiang, Q., Zhang, P., Cao, X., R. Khosravi, M. et al. (2022). Game theory for distributed iov task offloading with fuzzy neural network in edge computing. *IEEE Transactions on Fuzzy Systems*. DOI 10.1109/TFUZZ.2022.3158000.
12. Xu, X., Zhang, X., Liu, X., Jiang, J., Qi, L. et al. (2021). Adaptive computation offloading with edge for 5G-envisioned internet of connected vehicles. *IEEE Transactions on Intelligent Transportation Systems, 22(8),* 5213–5222. DOI 10.1109/TITS.2020.2982186.

13. Muniswamaiah, M., Agerwala, T., Tappert, C. C. (2021). A survey on cloudlets, mobile edge, and fog computing. *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, Washington DC, USA. DOI 10.1109/CSCloud-EdgeCom52276.2021.00034.
14. Joo, K. N., Youn, C. H. (2021). Accelerating distributed sgd with group hybrid parallelism. *IEEE Access, 9*, 52601–52618. DOI 10.1109/ACCESS.2021.3070012.
15. Huang, Y., Sun, S. Y., Duan, X, S., Chen, Z. G. (2016). A study on deep neural networks framework. *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1519–1522. Xi'an, China. DOI 10.1109/IMCEC.2016.7867471.
16. Hua, Y., Guo, J., Zhao, H. (2015). Deep belief networks and deep learning. *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pp. 1−4. Harbin, IEEE. DOI 10.1109/ICAIOT.2015.7111524.
17. Chen, Z., Yeo, C. K., Lee, B. S., Lau, C. T. (2018). Autoencoder-based network anomaly detection. *2018 Wireless Telecommunications Symposium (WTS)*, Phoenix, AZ, USA. DOI 10.1109/WTS.2018.8363930.
18. Albawi, S., Mohammed, T. A., Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, Antalya, Turkey. DOI 10.1109/ICEngTechnol.2017.8308186.
19. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H. et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. DOI 10.48550/arXiv.1406.1078.
20. Kim, K., Lee, D. J., Lim, H. K., Oh, S. W., Han, Y. H. (2021). Deep RNN-based network traffic classification scheme in edge computing system. *Computer Science and Information Systems, 19,* 165–184. DOI 10.2298/CSIS200424038K.
21. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B. et al. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine, 35(1),* 53–65.
22. Gui, J., Sun, Z., Wen, Y., Tao, D., Ye, J. (2021). A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE Transactions on Knowledge and Data Engineering*. DOI 10.1109/TKDE.2021.3130191.
23. Robbins, H., Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics, 22(3),* 400–407.
24. Sutskever, I., Martens, J., Dahl, G., Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, Atlanta, Georgia, USA, PMLR. https://proceedings.mlr.press/v28/sutskever13.html.
25. Dozat, T. (2016). Incorporating nesterov momentum into adam. https://5y1.org/info/keras-continue-training_4_53911b.html.
26. Ruder, S. (2016). An overview of gradient descent optimization algorithms. DOI 10.48550/arXiv.1609.04747.
27. Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. DOI 10.48550/arXiv.1412.6980.
28. Tran, P. T., Phong, L. T. (2019). On the convergence proof of amsgrad and a new version. *IEEE Access, 7,* 61706–61716. DOI 10.48550/arXiv.1904.03590.
29. De, S., Goldstein, T. (2016). Efficient distributed sgd with variance reduction. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, Spain. DOI 10.1109/ICDM.2016.0022.
30. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M. et al. (2012). Large scale distributed deep networks. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.), *Advances in neural information processing systems*, vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

31. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T. et al. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News, 45(1),* 615–629.

32. Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C. et al. (2018). A survey on the edge computing for the internet of things. *IEEE Access, 6,* 6900–6919. DOI 10.1109/ACCESS.2017.2778504.

33. Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K. et al. (2019). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE, 107(8),* 1738–1762. DOI 10.1109/JPROC.2019.2918951.

34. Xu, D., Li, T., Li, Y., Su, X., Tarkoma, S. et al. (2021). Edge intelligence: Empowering intelligence to the edge of network. *Proceedings of the IEEE, 109(11),* 1778–1837.

35. Caprolu, M., di Pietro, R., Lombardi, F., Raponi, S. (2019). Edge computing perspectives: Architectures, technologies, and open security issues. *2019 IEEE International Conference on Edge Computing (EDGE)*, Milan, Italy, IEEE. DOI 10.1109/EDGE.2019.00035.

36. Sabella, D., Vaillant, A., Kuure, P., Rauschenbach, U., Giust, F. (2016). Mobile-edge computing architecture: The role of mec in the internet of things. *IEEE Consumer Electronics Magazine, 5(4),* 84–91. DOI 10.1109/MCE.2016.2590118.

37. Biookaghazadeh, S., Zhao, M., Ren, F. (2018). Are {FPGAs} suitable for edge computing?. *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, United States.

38. Ren, J., Yu, G., He, Y., Li, G. Y. (2019). Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology, 68(5),* 5031–5044. DOI 10.1109/TVT.2019.2904244.

39. Li, Y., Xu, L. (2019). The service computational resource management strategy based on edge-cloud collaboration. *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China. DOI 10.1109/ICSESS47205.2019.9040830.

40. Lou, P., Liu, S., Hu, J., Li, R., Xiao, Z. et al. (2020). Intelligent machine tool based on edge-cloud collaboration. *IEEE Access, 8,* 139953–139965. DOI 10.1109/ACCESS.2020.3012829.

41. Tong, L., Li, Y., Gao, W. (2016). A hierarchical edge cloud architecture for mobile computing. *IEEE INFOCOM 2016–35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA. DOI 10.1109/INFOCOM.2016.7524340.

42. Vissicchio, S., Vanbever, L., Bonaventure, O. (2014). Opportunities and research challenges of hybrid software defined networks. *ACM SIGCOMM Computer Communication Review, 44(2),* 70–75. DOI 10.1145/2602204.2602216.

43. Xu, X., Wu, Q., Qi, L., Dou, W., Tsai, S. B. et al. (2021). Trust-aware service offloading for video surveillance in edge computing enabled internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems, 22(3),* 1787–1796. DOI 10.1109/TITS.2020.2995622.

44. Qi, L., Lin, W., Zhang, X., Dou, W., Xu, X. et al. (2022). A correlation graph based approach for personalized and compatible web apis recommendation in mobile app development. *IEEE Transactions on Knowledge and Data Engineering*. DOI 10.1109/TKDE.2022.3168611.

45. Xu, X., Shen, B., Yin, X., Khosravi, M. R., Wu, H. et al. (2021). Edge server quantification and placement for offloading social media services in industrial cognitive IoV. *IEEE Transactions on Industrial Informatics, 17(4),* 2910–2918. DOI 10.1109/TII.2020.2987994.

46. Li, D., Salonidis, T., Desai, N. V., Chuah, M. C. (2016). DeepCham: Collaborative edge-mediated adaptive deep learning for mobile object recognition. *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Washington DC, USA. DOI 10.1109/SEC.2016.38.

47. Wang, F., Zhang, C., Liu, J., Zhu, Y., Pang, H. et al. (2019). Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, Paris, France, IEEE. DOI 10.1109/INFOCOM.2019.8737456.

48. Zhang, C., Pang, H., Liu, J., Tang, S., Zhang, R. et al. (2019). Toward edge-assisted video content intelligent caching with long short-term memory learning. *IEEE Access, 7,* 152832–152846. DOI 10.1109/ACCESS.2019.2947067.

49. Hou, X., Ren, Z., Wang, J., Cheng, W., Ren, Y. et al. (2020). Reliable computation offloading for edge-computing-enabled software-defined iov. *IEEE Internet of Things Journal, 7(8),* 7097–7111. DOI 10.1109/JIOT.2020.2982292.

50. Tian, H., Xu, X., Qi, L., Zhang, X., Dou, W. et al. (2021). Copace: Edge computation offloading and caching for self-driving with deep reinforcement learning. *IEEE Transactions on Vehicular Technology, 70(12),* 13281–13293. DOI 10.1109/TVT.2021.3121096.

51. Chen, X., Ma, H., Wan, J., Li, B., Xia, T. (2017). Multi-view 3D object detection network for autonomous driving. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA. DOI 10.1109/CVPR.2017.691.

52. Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F. Y. (2014). Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems, 16(2),* 865–873.

53. Chu, T., Wang, J., Codecà, L., Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems, 21(3),* 1086–1095.

54. Alamgir Hossain, S., Anisur Rahman, M., Hossain, M. A. (2018). Edge computing framework for enabling situation awareness in IoT based smart city. *Journal of Parallel and Distributed Computing, 122,* 226–237. DOI 10.1016/j.jpdc.2018.08.009.

55. Shanmugapriya, P., Baskaran, J., Nayanatara, C., Kothari, D. P. (2019). IoT based approach in a power system network for optimizing distributed generation parameters. *Computer Modeling in Engineering & Sciences, 119(3),* 541–558. DOI 10.32604/cmes.2019.04074.

56. Wang, F., Gong, W., Liu, J. (2019). On spatial diversity in WiFi-based human activity recognition: A deep learning-based approach. *IEEE Internet of Things Journal, 6(2),* 2035–2047. DOI 10.1109/JIOT.2018.2871445.

57. He, Y., Mendis, G. J., Wei, J. (2017). Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism. *IEEE Transactions on Smart Grid, 8(5),* 2505–2516. DOI 10.1109/TSG.2017.2703842.

58. Li, L., Ota, K., Dong, M. (2018). Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics, 14(10),* 4665–4673. DOI 10.1109/TII.2018.2842821.

59. Dai, H., Xu, Y., Chen, G., Dou, W., Tian, C. et al. (2022). Rose: Robustly safe charging for wireless power transfer. *IEEE Transactions on Mobile Computing, 21(6),* 2180–2197. DOI 10.1109/TMC.2020.3032591.

60. Zhao, R., Yan, R., Wang, J., Mao, K. (2017). Learning to monitor machine health with convolutional bi-directional LSTM networks. *Sensors, 17(2),* 273.

61. Wang, F., Fan, X., Wang, F., Liu, J. (2019). Backup battery analysis and allocation against power outage for cellular base stations. *IEEE Transactions on Mobile Computing, 18(3),* 520–533. DOI 10.1109/TMC.2018.2842733.

62. Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S. et al. (2020). Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal, 7(8),* 7457–7469.

63. Elgendy, I. A., Muthanna, A., Hammoudeh, M., Shaiba, H., Unal, D. et al. (2021). Advanced deep learning for resource allocation and security aware data offloading in industrial mobile edge computing. *Big Data, 9(4),* 265–278.

64. Zhang, K., Zhu, Y., Leng, S., He, Y., Maharjan, S. et al. (2019). Deep learning empowered task offloading for mobile edge computing in urban informatics. *IEEE Internet of Things Journal, 6(5),* 7635–7647. DOI 10.1109/JIOT.2019.2903191.

65. He, Y., Sheng, B., Yin, H., Yan, D., Zhang, Y. (2022). Multi-objective deep reinforcement learning based time-frequency resource allocation for multi-beam satellite communications. *China Communications, 19(1),* 77–91. DOI 10.23919/JCC.2022.01.007.

66. Wang, F., Zhang, M., Wang, X., Ma, X., Liu, J. (2020). Deep learning for edge computing applications: A state-of-the-art survey. *IEEE Access, 8,* 58322–58336. DOI 10.1109/ACCESS.2020.2982411.

67. Naveen, S., Kounte, M. R. (2019). Key technologies and challenges in IoT edge computing. *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India. DOI 10.1109/I-SMAC47947.2019.9032541.

68. Wang, X., Han, Y., Leung, V. C., Niyato, D., Yan, X. et al. (2020). Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials, 22(2),* 869–904.

69. Chang, Z., Liu, S., Xiong, X., Cai, Z., Tu, G. (2021). A survey of recent advances in edge-computing-powered artificial intelligence of things. *IEEE Internet of Things Journal, 8(18),* 13849–13875. DOI 10.1109/JIOT.2021.3088875.

70. Lin, L., Liao, X., Jin, H., Li, P. (2019). Computation offloading toward edge computing. *Proceedings of the IEEE, 107(8),* 1584–1607.

71. Mach, P., Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials, 19(3),* 1628–1656.

72. Huai, Z., Ding, B., Wang, H., Geng, M., Zhang, L. (2019). Towards deep learning on resource-constrained robots: A crowdsourcing approach with model partition. *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBD-Com/IOP/SCI)*, Leicester, UK. DOI 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00194.

73. Ran, X., Chen, H., Zhu, X., Liu, Z., Chen, J. (2018). Deepdecision: A mobile deep learning framework for edge video analytics. *IEEE Conference on Computer Communications*, Honolulu, HI, USA. DOI 10.1109/INFOCOM.2018.8485905.

74. Han, S., Shen, H., Philipose, M., Agarwal, S., Wolman, A. et al. (2016). Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/2906388.2906396.

75. Chen, T. Y. H., Ravindranath, L., Deng, S., Bahl, P., Balakrishnan, H. (2015). Glimpse: Continuous, real-time object recognition on mobile devices. *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/2809695.2809711.

76. Habak, K., Ammar, M., Harras, K. A., Zegura, E. (2015). Femto clouds: Leveraging mobile devices to provide cloud service at the edge. *2015 IEEE 8th International Conference on Cloud Computing*, New York, NY, USA, IEEE. DOI 10.1109/CLOUD.2015.12.

77. Zhao, Z., Jiang, Z., Ling, N., Shuai, X., Xing, G. (2018). Ecrt: An edge computing system for real-time image-based object tracking. *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, Shenzhen, China. DOI 10.1145/3274783.3275199.

78. Matsubara, Y., Baidya, S., Callegaro, D., Levorato, M., Singh, S. (2019). Distilled split deep neural networks for edge-assisted real-time systems. *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, Los Cabos Mexico. DOI 10.1145/3349614.3356022.

79. Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T. et al. (2016). Federated learning: Strategies for improving communication efficiency. DOI 10.48550/arXiv.1610.05492.

80. Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C. et al. (2019). Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications, 37(6),* 1205–1221. DOI 10.1109/JSAC.2019.2904348.

81. Zhang, P., Sun, H., Situ, J., Jiang, C., Xie, D. (2021). Federated transfer learning for IIoT devices with low computing power based on blockchain and edge computing. *IEEE Access, 9,* 98630–98638.

82. Daga, H., Nicholson, P. K., Gavrilovska, A., Lugones, D. (2019). Cartel: A system for collaborative transfer learning at the edge. *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/3357223.3362708.

83. Choudhary, T., Mishra, V., Goswami, A., Sarangapani, J. (2020). A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review, 53(7),* 5113–5155.

84. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T. (2018). Rethinking the value of network pruning. DOI 10.48550/arXiv.1810.05270.

85. Polino, A., Pascanu, R., Alistarh, D. (2018). Model compression via distillation and quantization. DOI 10.48550/arXiv.1802.05668.

86. Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B. (2013). Low-rank matrix factorization for deep neural network training with high-dimensional output targets. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, IEEE. DOI 10.1109/ICASSP.2013.6638949.

87. Gou, J., Yu, B., Maybank, S. J., Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision, 129(6),* 1789–1819.

88. Mei, S., Chen, Y., H. Q. H. Y. D. L. B. S. L. Y. Y. L. (2022). A method based on knowledge distillation for fish school stress state recognition in intensive aquaculture. *Computer Modeling in Engineering & Sciences, 131(3),* 1315–1335. DOI 10.32604/cmes.2022.019378.

89. Agarwal, P., Alam, M. (2020). A lightweight deep learning model for human activity recognition on edge devices. *Procedia Computer Science, 167*, 2364–2373. DOI 10.1016/j.procs.2020.03.289.

90. Hu, B., Gao, B., Woo, W. L., Ruan, L., Jin, J. et al. (2020). A lightweight spatial and temporal multi-feature fusion network for defect detection. *IEEE Transactions on Image Processing, 30,* 472–486.

91. Yao, Y., Wang, Q., Mai, J., Yang, W. (2021). Crossnet: Computing-friendly lightweight anchor-free detector. *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, Chengdu, China, IEEE. DOI 10.48550/arXiv.2208.04622.

92. Zhang, Y. M., Lee, C. C., Hsieh, J. W., Fan, K. C. (2021). CSL-YOLO: A new lightweight object detection system for edge computing. DOI 10.48550/arXiv.2107.04829.

93. Sharma, A. K., Kang, B., Kim, K. K. (2021). Lightnet: A lightweight neural network for image classification. *2021 18th International SoC Design Conference (ISOCC)*, Jeju Island, Korea. DOI 10.1109/ISOCC53507.2021.9613865.

94. Jia, S., Lin, Z., Xu, M., Huang, Q., Zhou, J. et al. (2020). A lightweight convolutional neural network for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing, 59(5),* 4150–4163.

95. Kumar, A., Sharma, A., Bharti, V., Singh, A. K., Singh, S. K. et al. (2021). Mobihisnet: A lightweight cnn in mobile edge computing for histopathological image classification. *IEEE Internet of Things Journal, 8(24),* 17778–17789.

96. Tang, G., Zhao, L., Jiang, R., Zhang, X. (2019). Single image dehazing via lightweight multi-scale networks. *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, IEEE. DOI 10.1109/BigData47090.2019.9006075.

97. Zhang, Z., Jiang, Y., Jiang, J., Wang, X., Luo, P. et al. (2021). Star: A structure-aware lightweight transformer for real-time image enhancement. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Virtual. DOI 10.1109/ICCV48922.2021.00407.

98. Li, H., Cheng, Q., Li, X., Ma, S., Ma, J. (2021). Lightweight and fine-grained privacy-preserving data aggregation scheme in edge computing. *IEEE Systems Journal,* 1–10. DOI 10.1109/JSYST.2021.3112581.

99. Ma, Z., Ma, J., Miao, Y., Liu, X., Choo, K. K. R. et al. (2020). Lightweight privacy-preserving medical diagnosis in edge computing. *IEEE Transactions on Services Computing*. DOI 10.1109/TSC.2020.3004627.

100. Zhang, L., Gao, X., Mu, Y. (2020). Secure data sharing with lightweight computation in e-health. *IEEE Access, 8,* 209630–209643. DOI 10.1109/ACCESS.2020.3039866.

101. Fitwi, A. H., Chen, Y., Zhu, S. (2021). Enforcing privacy preservation on edge cameras using lightweight video frame scrambling. *IEEE Transactions on Services Computing*. DOI 10.1109/TSC.2021.3135352.

102. Chang, X., Li, W., Zomaya, A. Y. (2020). A lightweight short-term photovoltaic power prediction for edge computing. *IEEE Transactions on Green Communications and Networking, 4(4),* 946–955. DOI 10.1109/TGCN.2020.2996234.

103. Ren, L., Liu, Y., Wang, X., Lü, J., Deen, M. J. (2021). Cloud–edge-based lightweight temporal convolutional networks for remaining useful life prediction in iiot. *IEEE Internet of Things Journal, 8(16),* 12578–12587. DOI 10.1109/JIOT.2020.3008170.

104. Wang, S., Pu, Z., Li, Q., Guo, Y., Li, M. (2021). Edge computing-enabled crowd density estimation based on lightweight convolutional neural network. *2021 IEEE International Smart Cities Conference (ISC2)*, Manchester, UK, IEEE. DOI 10.1109/ISC253183.2021.9562877.

105. Zhao, Y., Deng, P., Liu, J., Wang, M., Wan, J. (2021). Lcanet: Lightweight context-aware attention networks for earthquake detection and phase-picking on IoT edge devices. *IEEE Systems Journal,* 1–12. DOI 10.1109/JSYST.2021.3114689.

106. Zhang, Z., Yin, L., Peng, Y., Li, D. (2018). A quick survey on large scale distributed deep learning systems. *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Singapore. DOI 10.1109/PADSW.2018.8644613.

107. Saputra, Y. M., Hoang, D. T., Nguyen, D. N., Dutkiewicz, E., Niyato, D. et al. (2019). Distributed deep learning at the edge: A novel proactive and cooperative caching framework for mobile edge networks. *IEEE Wireless Communications Letters, 8(4),* 1220–1223. DOI 10.1109/LWC.2019.2912365.

108. Thomas, D. B., Howes, L., Luk, W. (2009). A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/1508128.1508139.

109. Micikevicius, P. (2009). 3d finite difference computation on gpus using cuda. *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/1513895.1513905.

110. Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K. et al. (2013). More effective distributed ML via a stale synchronous parallel parameter server. In: Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. Q. (Eds.), *Advances in neural information processing systems*, vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/b7bb35b9c6ca2aee2df08cf09d7016c2-Paper.pdf.

111. Collins, A., Grewe, D., Grover, V., Lee, S., Susnea, A. (2014). Nova: A functional language for data parallelism. *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY'14, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/2627373.2627375.

112. Guide, D. (2013). Cuda c programming guide. In: *NVIDIA*. https://eva.fing.edu.uy/pluginfile.php/174141/mod_resource/content/1/CUDA_C_Programming_Guide.pdf.

113. Munshi, A. (2009). The opencl specification. *2009 IEEE Hot Chips 21 Symposium (HCS)*, Stanford, CA, USA, IEEE.

114. Intel, I. (2009). *Threading building blocks reference manual*.

115. Milner, R. (1978). A theory of type polymorphism in programming. *Journal of Computer and System Sciences, 17(3),* 348–375. DOI 10.1016/0022-0000(78)90014-4.

116. Li, H., Kadav, A., Kruus, E., Ungureanu, C. (2015). Malt: Distributed data-parallelism for existing ML applications. *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/2741948.2741965.

117. Zhang, Y., Liao, X., Gu, L., Jin, H., Hu, K. et al. (2020). Asyngraph: Maximizing data parallelism for efficient iterative graph processing on GPUs. *ACM Transactions on Architecture and Code Optimization, 17(4)*. DOI 10.1145/3416495.

118. Chen, C. C., Yang, C. L., Cheng, H. Y. (2018). Efficient and robust parallel DNN training through model parallelism on multi-GPU platform. DOI 10.48550/arXiv.1809.02839.

119. Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N. R. et al. (2018). Pipedream: Fast and efficient pipeline parallel DNN training. DOI 10.48550/arXiv.1806.03377.

120. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J. et al. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. DOI 10.48550/arXiv.1909.08053.

121. Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J. et al. (2018). Gpipe: Efficient training of giant neural networks using pipeline parallelism. DOI 10.48550/arXiv.1811.06965.

122. Jia, Z., Zaharia, M., Aiken, A. (2019). Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, vol. 1. DOI 10.48550/arXiv.1807.05358.

123. Zhu, W., Zhao, C., Li, W., Roth, H., Xu, Z. et al. (2020). Lamp: Large deep nets with automated model parallelism for image segmentation. In: Martel, A. L., Abolmaesumi, P., Stoyanov, D., Mateus, D., Zuluaga, M. A. et al. (Eds.), *Medical image computing and computer assisted intervention–MICCAI 2020*. Cham: Springer International Publishing. DOI 10.48550/arXiv.2006.12575.

124. Jain, A., Awan, A. A., Aljuhani, A. M., Hashmi, J. M., Anthony, Q. G. et al. (2020). GEMS: GPU-enabled memory-aware model-parallelism system for distributed dnn training. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Atlanta, GA, USA. DOI 10.1109/SC41405.2020.00049.

125. Park, J. H., Yun, G., Yi, C. M., Nguyen, N. T., Lee, S. et al. (2020). HetPipe: Enabling large DNN training on (whimpy) heterogeneous GPU clusters through integration of pipelined model parallelism and data parallelism. *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association. https://www.usenix.org/conference/atc20/presentation/park.

126. Pal, S., Ebrahimi, E., Zulfiqar, A., Fu, Y., Zhang, V. et al. (2019). Optimizing multi-GPU parallelization strategies for deep learning training. *IEEE Micro, 39(5),* 91–101. DOI 10.1109/MM.2019.2935967.

127. Mirhoseini, A., Pham, H., Le, Q. V., Steiner, B., Larsen, R. et al. (2017). Device placement optimization with reinforcement learning. *Proceedings of the 34th International Conference on Machine Learning. Volume 70 of Proceedings of Machine Learning Research*, PMLR. DOI 10.48550/arXiv.1706.04972.

128. Wu, L., Zhu, Z., E, W. (2017). Towards understanding generalization of deep learning: Perspective of loss landscapes. DOI 10.48550/arXiv.1706.10239.

129. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *5th International Conference on Learning Representations*. http://arxiv.org/abs/1609.04836.

130. Chen, M., Wang, T., Zhang, S., Liu, A. (2021). Deep reinforcement learning for computation offloading in mobile edge computing environment. *Computer Communications, 175,* 1–12.

131. Chen, M., Liu, W., Wang, T., Liu, A., Zeng, Z. (2021). Edge intelligence computing for mobile augmented reality with deep reinforcement learning approach. *Computer Networks, 195*, 108186.

132. Chen, M., Liu, W., Wang, T., Zhang, S., Liu, A. (2022). A game-based deep reinforcement learning approach for energy-efficient computation in mec systems. *Knowledge-Based Systems, 235,* 107660. DOI 10.1016/j.knosys.2021.107660.

133. Stahl, R., Hoffman, A., Mueller-Gritschneder, D., Gerstlauer, A., Schlichtmann, U. (2021). DeeperThings: Fully distributed cnn inference on resource-constrained edge devices. *International Journal of Parallel Programming, 49(4),* 600–624.

134. Zhao, Z., Barijough, K. M., Gerstlauer, A. (2018). DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11),* 2348–2359. DOI 10.1109/TCAD.2018.2858384.
135. Hu, Z., Tarakji, A. B., Raheja, V., Phillips, C., Wang, T. et al. (2019). Deephome: Distributed inference with heterogeneous devices in the edge. *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, EMDL '19, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/3325413.3329787.
136. Liu, D., Chen, X., Zhou, Z., Ling, Q. (2020). Hiertrain: Fast hierarchical edge ai learning with hybrid parallelism in mobile-edge-cloud computing. *IEEE Open Journal of the Communications Society, 1,* 634–645. DOI 10.1109/OJCOMS.2020.2994737.
137. Kozik, R., Choraś, M., Ficco, M., Palmieri, F. (2018). A scalable distributed machine learning approach for attack detection in edge computing environments. *Journal of Parallel and Distributed Computing, 119,* 18–26. DOI 10.1016/j.jpdc.2018.03.006.
138. Moreno-Alvarez, S., Haut, J. M., Paoletti, M. E., Rico-Gallego, J. A. (2021). Heterogeneous model parallelism for deep neural networks. *Neurocomputing, 441,* 1–12. DOI 10.1016/j.neucom.2021.01.125.
139. Amatya, V., Vishnu, A., Siegel, C., Daily, J. (2017). What does fault tolerant deep learning need from mpi? *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, New York, NY, USA, Association for Computing Machinery. DOI 10.1145/3127024.3127037.
140. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T. et al. (2020). A survey on distributed machine learning. *ACM Computing Surveys, 53(2),* 1—33. DOI 10.1145/3377454.