# Edge Computing and Distributed Intelligence

Yuansong Qiao, Abdelhakim Senhaji Hafid, Nazim Agoulmine, Amir Karamoozian, Lotfi Tamazirt, Brian Lee

## Table of Contents

## List of Abbreviations

| Abbreviation | Expansion |
|---|---|
| AI | Artificial Intelligence |
| Async-SGD | Asynchronous Stochastic Gradient Descent |
| CNN | Convolutional Neural Networks |
| CoAP | Constrained Application Protocol |
| CSA | Connected Streaming Analytics |
| CVT | Container-based Virtualization Technology |
| DAI | Distributed Artificial Intelligence |
| DBM | Deep Boltzmann Machines |
| DBN | Deep Belief Networks |
| DNN | Deep Neural Networks |
| GPU | Graphics Processing Units |
| IoT | Internet of Things |
| MPI | Message Passing Interface |
| MQTT | Message Queuing Telemetry Transport |
| ODP | Optimal Distributed Stream Processing Problem |
| QoS | Quality of Service |
| RNN | Recurrent Neural Networks |
| SaaS | Software as a Service |
| SDN | Software Defined Network |
| SGD | Stochastic Gradient Descent |
| Sync-SGD | Synchronous Stochastic Gradient Descent |
| VM | Virtual Machine |

## Abstract

The convergence of Internet of Things (IoT), Artificial Intelligence (AI), 5G, Big Data and cloud computing technologies is creating an emerging edge to cloud computing model to overcome the challenges of traditional cloud computing in supporting the upcoming Internet of (smart and autonomous) things. Edge computing complements cloud computing by offering local computational facilities to enable processing of extremely large amount of data originated from IoT devices and providing intelligent decisions close to the things. As edge devices are geographically dispersed and may be resource constraint, many challenges exist in deploying and executing intelligent applications over these heterogeneous resources. This chapter will address these challenges through 3 sections to discuss: 1) what is edge computing and the resource management approaches in edge environments, especially application deployment mechanisms in the IoT-Edge-Cloud continuum; 2) the benefits and challenges of implementing distributed intelligence and data analytics in the edge environments with exemplary use cases; 3) the distributed learning architecture and algorithms for edge computing, with a special focus on distributed deep learning based approaches.

## Keywords

Edge Computing, Fog Computing, Resource Management, Resource Orchestration, Edge Intelligence, Distributed Deep Learning, Parallelism, Distributed AI, Embedded Devices, Resource Constraint.

# 1 Edge Computing and Resource Management

## 1.1 Introduction

With the emergence of Internet of Things (IoT), where any device, however small, is able to connect to the Internet and monitor/control physical elements, many applications were made possible such as smart cities, smart homes, smart healthcare and smart transportation. IoT connections are established by resource-constrained edge devices which connect to existing network infrastructures. This enables an IoT-to-cloud architecture in which the infrastructure between device and cloud is only used as a communication platform. Current cloud computing helps us to store and process bulk volumes of data from battery drained devices; however, with the rate, distribution and scale of extraordinary data from IoT devices, processing all the data in the cloud resources would be impractical. However, not every stream analytic is of the same importance or needs the same requirement, and by processing streams closer to the proximity of stream sources (i.e. IoT devices), it leaves us a chance to make more intelligent trade-offs among latency, bandwidth and data privacy. For instance, in the case of IoT stringent latency requirements, or geographically constrained smart IoT devices, cloud computing cannot fully address these requirements; or as another example, in the case of health care, for the sake of patient safety, we can't fully rely on remote cloud resources because of the possibility of network and/or cloud real time resource request failures. Generally, the current state of existing cloud infrastructure is limited to only two unsatisfactory actions: either (1) offload and process all input IoT data in the cloud which cause additional communication costs and latency, or (2) process all data locally which is not possible especially in case of applications with high-computational overhead. More specifically, to support IoT applications, several challenges must be overcome [1] [2] [3]: (1) Latency: Generally, most IoT applications have stringent latency requirements that cloud computing cannot adequately satisfy. For example, road safety and self-driving car services require latencies of less than 50ms while smart factories have even stricter requirements, with latencies ranging from 250µs to 10ms. Latency incurred to send data to the cloud, for processing via the core network, may cause unacceptable latency; (2) bandwidth: rapid increase in the number of 'smart things' producing data within the next few years, as projected in ABI Research in 2015, shows that data captured by IoT devices will exceed 1.6 zettabytes by 2020 (e.g., hundreds of terabytes per day in the case of smart factories, or gigabytes of data per minute in the case of self-driving cars). Sending all this data to the cloud would cause excessive stress on the backhaul communication links; (3) Privacy and security: There are a lot of use cases where IoT devices generate a collection of sensitive data (e.g., end user identity, location or healthcare data) that requires more careful privacy

preservation, if transmitted over the public Internet to the cloud [4]. Existing cloud privacy and security measurements cannot be directly applied to IoT-Cloud continuum due to its distinct characteristics, such as large scale geo-distribution, heterogeneity or mobility. Scaling out in geographical span leads to no centralized control over the data travelling from the IoT device to the Cloud, and relying merely on resource-constrained IoT devices for data encryption/decryption is not possible due to insufficient IoT device resources; thus, new privacy, confidentiality, integrity and availability challenges arise; and (4) Context awareness: Context is formally defined as "any information that can be used to characterize the situation of an entity" [5]. Context information in IoT era can particularly include, for instance, information about a set of nearby devices, or local network status. Physical distance between Cloud and IoT devices results in lack of proximity which, in turn, causes only limited context to be shared between Cloud and IoT. For example, let us consider a cloud-based service that detects car accidents in roads or at intersections. Due to the lack of local context information, that service won't be able to disseminate alerts to other vehicles in the vicinity of the accident. Therefore, there is a need to design a new computing paradigm to address these challenges.

## 1.2 Towards Edge/Fog Computing

To overcome cloud deficiencies, a new paradigm, called fog computing has emerged [6]. As fog computing and edge computing are both widely used by the community to denote a similar concept, this chapter uses the two terms interchangeably. This new paradigm has been introduced as an extension of cloud computing paradigm; it moves the computing (and storage) resources to the proximity of devices/users from the core to the edge of the network (see Figure 1-1). The objective is to support wide range of IoT applications, and thus processing data anywhere along the IoT-Fog (Edge)-Cloud continuum. This distributed computing structure has inherent advantages; it also provides system scalability which allows the infrastructure to support processing of large-scale data generated by geographically distributed IoT devices. With fog computing, processing of the data is performed closer to the edge, and only digests or exceptions are sent into the cloud for more processing or storage.

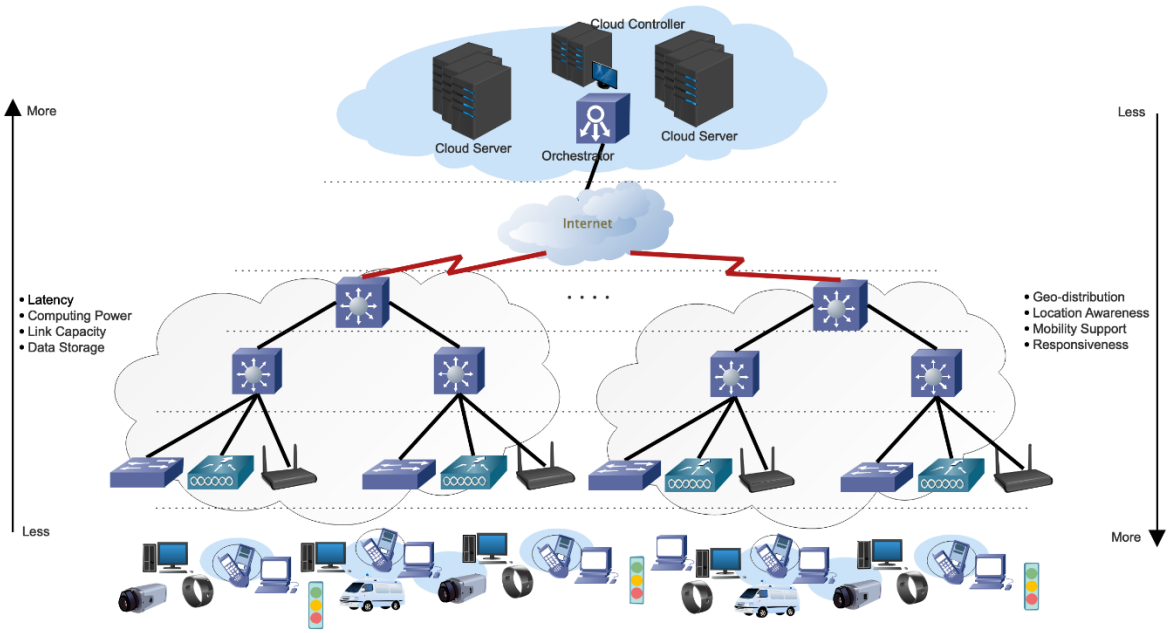



***Figure 1-1: IoT-Fog-Cloud Architecture***

It is a platform placed between traditional cloud infrastructure and the things, as a bridge, to provide storage, computation and networking capacities (Figure 1-1). It is considered as complementary to the cloud, addressing emerging IoT applications that require low latency and/or fast mobility support. A

common example that is often used to distinguish fog and cloud computing is whether ultra-low latency applications can be supported while maintaining satisfactory quality of service (QoS). Fog computing enables organizations to process inordinate amount of real-time and low-latency IoT data locally at the edge without consuming the organization's network bandwidth to send all the data back and forth to the cloud. Furthermore, by analysing and processing data closer to the source, Fog Computing can significantly improve the application performance.

The main difference between cloud and fog computing is also regarded in term of scale of hardware components associated with these computing paradigms. Cloud computing provides high availability of computing resources at relatively high power consumption, while fog computing provides moderate availability of computing resources at lower power consumption. Cloud computing typically utilizes large data centres, whereas fog computing uses small servers, routers, switches, gateways, and/or access points. Fog computing can be accessed through connected devices from the edge of the network, whereas cloud computing must be accessed through the wide area network core. Moreover, continuous Internet connectivity is not essential for the fog-based services to work. That is, the services can work independently with low or no Internet connectivity and send necessary updates to the cloud whenever the connection is available. Cloud computing, on the other hand, requires devices to be connected when the cloud service is in progress.

However, real-time data processing frameworks at the edge of the network is still an open research area [7]. Edge computing is indeed still at its infancy and many challenges have to be addressed. Despite, newly introduced commercial systems such as Azure IoT edge and AWS Greengrass, they still do not integrate mechanisms to recover from failure, nor do they include to dynamically distribute processing across devices, edge equipment and cloud infrastructure and coordinate it efficiently, as needed. Indeed, finding the best place(s) in the fog layer corresponding to a particular processing service (e.g., data trimming, face recognition) is a difficult problem. There is always a challenge to determine ideal trade-off between computing costs and transmission costs for an optimal placement while satisfying the service requirements (e.g., delay). This challenge is exacerbated due to the uncertain behaviour of the fog layer especially communication links connecting fog nodes to fog nodes, fog nodes to the cloud, and IoT devices to the fog nodes. Intuitively, one may say that by placing the service components as close as possible to the edge devices we can reduce transmission costs. However, fog nodes close to the edge are likely to have limited processing and storage capabilities. In addition, these nodes are often not as robust or well-maintained as centralized cloud resources, and solutions instilling resilience to failure are required. So they may not be capable of hosting sophisticated services. Furthermore, the service components may have different hardware and/or software requirements; fog nodes are also heterogeneous in terms of their hardware and software capabilities. Finally, with the characteristics of edge computing arise also new security challenges where existing mechanisms for cloud computing cannot be directly applied [8]. Some of the challenging security issues in edge computing include managing trust. Trust plays an important role in edge computing because processing is no longer centralized in the cloud. Solutions need to be devised to verify that devices where processing is performed are indeed genuine. Achieving trust in this context requires a robust trust model in place, a topic that has been barely investigated in the literature.

## 1.3 Resource Management in Edge/Fog Computing

The resource management problem that has been studied the most in the literature is resource optimization; the objective is to find the best place(s) in the IoT-Fog-Cloud continuum to process data, generated by IoT devices. The definition of the term 'best' varies depending on the considered system. Generally, it means allocating the required resources to process data that maximizes resource utilization while satisfying the requirements (e.g., delay) of IoT applications.

Yu et al. [9] proposed an optimization model to jointly study service placement and data routing in the fog. However, they focus only on fog tier and not the IoT-Fog-Cloud continuum. Similarly, in [10] and

its extension [11], the authors proposed a simulation-based approach to optimize IoT service placement in the fog. However, these approaches [9] [10] [11] do consider an IoT service as a single task which is not the case for several IoT services that are composed of more than one task. Furthermore, the approaches in [10] [11] support only batch processing of service requests. Gu et al. [12] investigated a fog-based resource management scheme for medical cyber-physical systems taking into account base station association, task distribution, and VM placement, aiming at minimizing the communication and VM deployment costs. However, they focus only on the fog tier which consists of only base stations.

Cao et al. [13] realized actual edge computing systems to collect and analyse IoT data (related to patients' fall detection), in order to achieve critical monitoring. However, in their implementation, sensitive real-time data processing is statically assigned to specific edge equipment without a particular attention dedicated to resilience to changes (e.g., overload and failure). Perera et al. [14] placed their computation containers on a single node at leaf-tier of the fog for pre-processing regardless of its burden on the fog node. Gia et al. [15] made use of two or more edge nodes on a direct path between IoT devices and the cloud without a specific coordination between processing at the edge and cloud. An example is described by López et al. [16], in which they have developed shirts with embedded sensors that collect physiological data about the patient and also act as fog gateways. However, it has been shown that coordination between edge computing and cloud computing is necessary for efficient deployment [17].

Benoit et al. [18] demonstrated that scheduling linear chains of processing operators onto a cluster of heterogeneous computing resources is an NP-Hard problem. Shen et al. [19] proposed Cisco's Connected Streaming Analytics (CSA) in which an architecture is provided to efficiently handle data stream processing queries for IoT applications by exploiting data centre and edge computing resources. The authors in [20] analysed the placement problem for a subset of distributed streaming application topologies, i.e., serial parallel decomposable graphs. This allowed them to exploit strong theoretical foundations and proposed an approximation algorithm, which, however, could allocate processing elements only on resources with uniform capacity. Huang et al. [21] elegantly rearranged the Multi-Constrained Optimal Path problem to model and solve the service composition problem. Nevertheless, their model allows only to express constraints on communication links but not on computing resources. Cardellini et al. [22] introduced an integer programming formulation taking into account resource heterogeneity for the Optimal Distributed Stream Processing Problem (ODP). But this approach is less general since it is not tied up to both correlated and uncorrelated processing elements. Mayer et al. [23] proposed EmuFog which is an emulation framework tailored for fog computing scenario that enables the emulation of real applications and workloads. However, they assume the existence of the communication topology and use very simplistic greedy algorithms to place fog nodes.

In summary, there have been several contributions in resource management and particularly resource optimization. However, there is not a single contribution that covers all issues, at the same time, raised by fog computing including uncertainty, resiliency, reliability, heterogeneity and mobility. Furthermore, there is not work on fog federation where one can control and federate fog resources across multiple operating domains. Finally, all contributions assume the existence of the fog infrastructure and devise schemes/models based on this assumption. It will be interesting to develop a unified model for the fog design problem that takes into account all the parameters that have a significant impact on the infrastructure, the requirements of the fog service provider (e.g., expected amount of load/IoT devices/requests), the constraints of the physical environment (e.g., potential locations of routers/fog nodes), and the cost.

# 2 Benefits and Challenges of Edge Intelligence

## 2.1 Introduction

The large flow of unstructured data generated by a large variety of possible connected devices with various capacities has allowed the emergence of several IoT architectures. The common properties of these architectures are the capacities of transportation of the generated data to centralized platforms deployed either in backend systems or in the cloud. The advantage of cloud computing is to provide on-demand resources to achieve scalability in processing the big data collected from potentially billions of connected objects. In addition, Cloud computing permits to execute advanced data analytics as well as machine learning algorithms that are more and more provided as SaaS (Software as a Service) easing the conception, deployment and execution. However, centralizing the transportation of all the data and processing can be very costly consuming uselessly a lot of resources that could be saved if the processing is pushed to the boundary of the network if this is possible. Hence, for some services and applications that require very low latency, the centralized processing in the cloud represents a downside that is difficult to address. Thanks to Fog Computing technology, we can execute advanced data analytics and AI algorithms at the edge and close to the objects that generate the data [24]. This section aims to present the two approaches and compare them in terms of advantages and weakness as well as the scientific and technological challenges that are being addressed in distributed AI in the edge of the network.

## 2.2 IoT Data Analytics and AI Analytics in the Cloud

The large flow of unstructured data generated by a large variety of possible connected devices with various capacities has allowed the emergence of several IoT architectures. The common properties of these architectures are the capacities to transport of the generated data to centralized platforms, using standard protocols such MQTT and CoAP, store, process and analyse this data in these centralized platforms. To achieve scalability in processing the huge data that can be collected from remote objects, these platforms are deployed in cloud computing infrastructures and get benefits from the elasticity of available resources (processing, storing and communication). These centralized cloud computing infrastructures have been enabled by the huge advances in virtualization technologies and quickly adopted by IoT players motivated by the potential reduction of cost (vs deploying their own backend infrastructure) and time to market to deploy their new services. Hence, many cloud providers have provided a plethora of advanced data analytics and machine learning algorithms along with data streaming and storing services SaaS (Software as a Service) that can be used on-demand and as per-use billing by IoT services providers to design new IoT oriented intelligent services.

## 2.3 AI Analytics from Cloud Computing to Edge Computing

While this centralized approach is today mainstream approach, it introduces several drawbacks such a large response time not complying with some real-time applications constraints and huge communication resources usage since all the data need to be transported to the datacentres that are usually far away from the connected devices. Hence, the storage of the collected data in datacentres that can be located in different regions and countries raises numerous problems of security and privacy. Therefore, it appears that pushing data analytics and intelligent algorithms closer to IoT objects could constitute a very promising approach.

Edge computing technology has made this possible since it provides computing infrastructure at the edge of the network where advanced data analytics and AI algorithms can be deployed to process data closer to it sources. This approach permits to reduce latency between data collection and processing/decision. It allows to resolve some problems of security and privacy of the collected data since it could remain closer to the sources (i.e. same country and regulation). In addition, real-time data processing, made possible by edge computing, opens up new possibilities. This approach permits therefore to distribute the intelligence in the network as close as possible to the connected objects

[25] and only transport important data and decision to the centralized system reducing not only the network resources consumption but also the latency of the decisions.

This section will discuss how IoT platform has moved from stand-alone platforms into cloud based platforms integrating advance data analytics and AI algorithms. It then presents the limitations of this approach and how distribution of the intelligence closer to the connected objects could help improving the performances of these platforms, thanks to edge computing that provides the necessary computing/storage and communication resources at the boundaries of the network (Figure 2-1) [26]. AI-powered edge computing will eventually help to increase operational efficiency with faster and more accurate predictions that will allow to reduce operation time and improve efficiency.

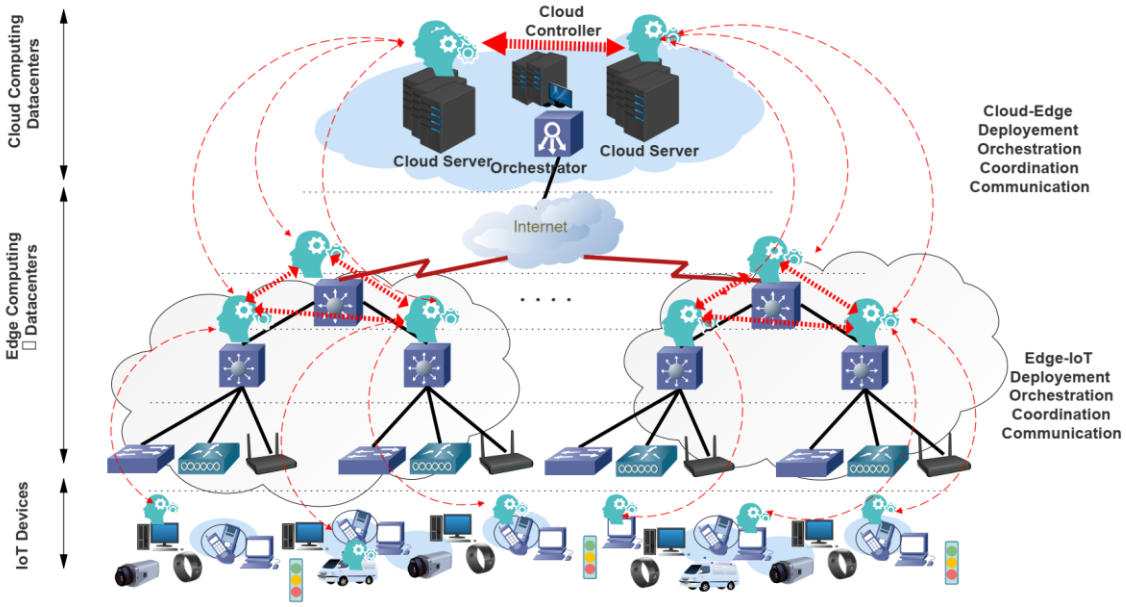



***Figure 2-1: Data Analytics from Edge to Cloud***

Since the intelligence is executed in a distributed manner close IoT devices and not executing it in a central place, there are several advantages to this approach [26] [27] as shown in Table 2-1.

***Table 2-1: Benefits of Edge Enhanced AI***

| Value properties | Description |
|---|---|
| **Lower response time** | IoT data processing is executed at the edge of the network and closer to the devices that transmit them reducing therefore response time of decision-making |
| **Higher privacy and security** | Data transmitted by IoT devices are not transported over large distance as in the central approach where it could be transported possibly across several administrative domains (including public-Internet) exposing it to several risks of security and privacy leakage issues. IoT devices and edge computing nodes will be in close geographic area reducing the risks and allowing higher control. |

| | |
|---|---|
| **Self-organized** | Intelligence in edge computing will allow standalone systems to take local decisions but also to collaborate with each other to allow the emergence of global Intelligence without the need of human operators. |
| **Higher flexibility** | Distributed AI from central cloud computing to edge computing will allow the emergence of innovative solutions that are flexible and sensitive to context. AI executed in the edge permits to take more efficiently into account situations and specificities (e.g. location, cost, capabilities, etc.) of the target system to control. |

## 2.4 Technical and Scientific Challenges

While there are numerous benefits of deploying edge intelligence, the distribution of intelligence and decision-making among numerous edge computing points raises many challenges that must be overcome efficiently [28]. These challenges can be grouped into two parts that are (1) deployment models and (2) distributed intelligence.

The first challenge, named deployment models is related to the deployment, orchestration and management of edge nodes to provide the necessary resources (computing, storage and communication) between IoT devices and cloud data centres. The deployed edge nodes consist of micro data centres that will be potentially deployed at large and be able to execute AI programs providing them necessary on-demand elastic local high computing, storage and communication capacities. Efficient and energy aware communication capabilities with IoT network gateways and/or IoT devices are necessary to collect data that is required for decision-making. Technologies such SDN (Software Defined Network) and virtualization are mandatory to instrument efficiently these edge data centres and execute AI based programs in a secure and isolated manner. Furthermore, container-based virtualization technology (CVT) will also constitute a key-element to allow flexibility in the control and orchestration of the infrastructure from IoT devices to the cloud via the edge.

The second group of challenges is related to solving the problem of convergence of distributed AI. Indeed, while distributing machine learning among numerous points of decision, there is a need to achieve not only local decisions but also global decisions. For that, there is a need to define a common knowledge model that can be used by distributed edge computing nodes in order to model and exchange knowledge that can be used by individual edge computing nodes to infer global decisions [29]. There is a need in particular to parallelize/distribute AI algorithms (e.g. Deep Learning algorithms) [30] [31]. In this case, the objective is to define new strategies to split AI models across different edge micro data centres as well duplication of data across these centres to train the underlying neural networks [32].

## 2.5 Use Cases of Distributed AI in the IoT-Edge-Cloud Continuum

***Industry 4.0***

In the industrial sector, improving productivity is a key element. Edge computing can contribute to the maximization of the productivity as it will transform the collected data from a large set of connected sensors that monitor the industrial process into valuable data which will help to improve the value chain as well as avoid accidents and predict failures [33]. The fast decisions that will be taken in the edge instead of the far end cloud will allow triggering real-time actions and messages to change or update the industrial process using connected actuators. Before reaching this objective, it is however necessary to fill the technological gap to allow the safe deployment of DAI (Distributed Artificial Intelligence) in the Edge. In particular, it is necessary to verify the authenticity and the credibility of the received data from IoT devices as it constitutes the input of the analysis and the decision making.

Moreover, edge computing ecosystems have to be prepared in order to deploy more applications, but also for the optimization of system operation. For this, management of secure identity has to be implemented to ensure that third parties access safely to the communications. This is also true for machine learning technologies that will contribute to the automation of the operations. Machine learning algorithms should be tested and validated in simulation environments before their deployment in the edge.

### *Workers Safety*

For worker monitoring, an intelligent gateway is often used to collect positions / orientation / state of each employee through modules that are tasked to "learn" the normal behaviour of the employee [34] and be able in case of anomaly / attack or behaviour change to trigger an alarm. However, in this case, some false alarms often distort the decision making, interpreting a normal movement as an anomaly. Indeed, this is due to the fact that the processing power of the modules is not sufficient to distinguish between a normal activity and an exceptional situation. Besides, the backhaul mechanism for communication plays an important role in the routing of data to the monitoring centre.

### *Urban IoT/Smart Cities*

Edge intelligence helps community leaders deploy smart urban projects to adapt to growing cities while reducing management costs. For this purpose, different IoT gateways, which have been designed by Huawei and Dell using the Niagara framework [35] developed by Tridium, aim to provide a commercial solution to instrument large IoT Networks and related generated data. The framework allows to collect and analyse building data in order to detect problems, carry out preventive maintenance and entrust decision-making tools to managers. From the gateways, it will also be possible to manage the temperature, quality of air and consumption of energy resources. However, in this kind of applications, some issues have to be addressed as the interaction between the different IoT modules, the control and orchestration, and the synchronization of different context.

### *Indoor Localization/Tracking*

Indoor Localization includes many potential fields of application: medical, military, healthcare, etc. In all these domains, latency is considered as a real issue since the receiver can change its position at any time. To solve this problem, it is necessary to process localization data as close as possible to the tracked object. In order to achieve high accuracy and response time, edge computing combined with AI can be used to provide the necessary trusted installation environment to host localization algorithms (possibility as containers) at the edge of the network, provide the communication environment to communicate directly via gateways with the IoT objects that generate the raw localisation data [36].

### *Smart Agriculture*

From drone data collection to connected automatic sprinklers, to numerous stand-alone sensors placed in plots, the agricultural sector is undergoing a major technological transformation. From cloud to connected objects, tools and practices have been evolving to increase the efficiency of the industry's productivity. Indeed, IoT is proving to be a way to optimize production and increase revenue while applying sustainable methods. Microsoft Azure and Schneider propose with Waterforce an IoT solution for measuring water consumption. Waterforce is developing SCADAFarm [37], a solution for remote control of plot watering devices and the supply of water points for livestock. The tool makes it possible to obtain the pressure in the equipment, to monitor the flows and detect the leaks, to know available water reserves, humidity rate of parcels and to receive alerts in case of breakdowns. Although the cost of installation is not an obstacle for the deployment of the solution, the establishment of the system requires a wide area communication network to transport all the data to backend cloud based application which make operational cost higher. Intelligence in the Edge

computing will certainly improve the solution taking decisions close to the farm while benefiting from global decisions taken in the cloud computing in coordination with the various Edges.

# 3 Distributed Intelligence at the Edge

## 3.1 Introduction

This section introduces the technologies and challenges for implementing distributed learning system in hybrid cloud / edge environments using deep learning technologies. It starts by introducing the background knowledge for distributed training and inference approaches in deep learning algorithms, including deep learning parallelism approaches (data parallelism, model parallelism, and combination of the both) and frameworks used in distributed training (i.e. parameter server). It continues to introduce various technologies that can be employed to run deep neural networks in resource constraint edge environments, including various lightweight deep neural network models, federated learning, data parallelism and model parallelism for edge environments.

## 3.2 Distributed Deep Learning Models and Architectures

Deep learning refers to a class of machine learning technologies which are composed of multiple layers of non-linear operations for pattern classification and feature or representation learning, e.g. Convolutional Neural Networks (CNN) [38], Deep Belief Networks (DBN) [39], Deep Boltzmann Machines (DBM) [40], and Recurrent Neural Networks (RNN) [41]. Deep learning has achieved superior performance in many application fields, e.g. speech recognition, image and video classification, and language translation. It outperforms traditional shallow machine learning architecture as it can more efficiently represent non-linear functions, which means a significantly large (deep) architecture is required to compactly represent non-linear functions [42]. Research shows that the accuracy of deep learning can be significantly improved by increasing the learning scale in terms of the training dataset size and/or the model size [43] [44]. Realistic deep model training involves a large volume of training data (from 1TB to 1PB) and parameters ($10^9$ to $10^{12}$) [45]. Consequently, scaling up deep learning algorithms has become a key research focus in the community. Utilising GPUs (Graphics Processing Units) as a parallel method to train Deep Neural Networks (DNN) is a significant achievement in recent years to tackle modestly sized deep networks in practice [46]. However, the training speed-up is limited when the model cannot fit in the GPU memory [43].

The trend is to implement the parallelism on a cluster of machines [43] [47]. These methods can be classified as data parallelism, model parallelism, or a mixture of both [48]. In data parallelism, the whole training dataset is divided into partitions. Each worker machine has a complete copy of the DNN model and performs training computation on the partitions assigned to it. In model parallelism, each machine computes on different parts of a single DNN model. The two parallelism models can be combined together, e.g. using model parallelism across GPUs on one machine and using data parallelism across machines. Currently, data parallelism is the most widely accepted scheme in practice. Pipeline parallelism is an emerging scheme by combining data and model parallelism [49] [50]. During the training process of these parallelism schemes, parameters generated by each worker need to be exchanged or synchronized, which leads to different parameter exchanging architectures, e.g. parameter servers [45] and Peer-to-Peer based approaches. The sections below will present more details on parallelism models and parameter exchange architectures.

### 3.2.1 Parallelism Models

***Basic Concept in DNN Training***

A DNN model normally consists of one input layer, one output layer and a number of hidden layers and each layer consists of a number of artificial neurons. Each neuron generates an output signal by transforming input signals from other neurons and/or itself. This is achieved by feeding a weighted sum of input signals into an activation function.

A DNN model training normally consists of multiple epochs. Each epoch is an iteration that goes through all training samples in the dataset. One epoch contains multiple training iterations and each of them performs a training over a small portion of the whole dataset, called mini-batch. The DNN training process contains two major steps, i.e. forward and backward propagations. In the forward propagation step, the input dataset passes through DNN layers from the input layer to the output layer and generates output signals, i.e. the predicted results based on the input dataset. Prediction errors are calculated through a loss function taking the predicted results and the desired results as augments. In the backward propagation step, gradients of the loss function with respect to the model parameters are calculated from the output layer to the input layer through the chain rule of differentiation. A widely adopted parameter updating method is Stochastic Gradient Descent (SGD) which calculates a stochastic estimation of the gradient. In normal gradient descent, the gradient is calculated based on the summed errors of all input samples, whereas in SGD, the estimated gradient is calculated from a random selected data sample or mini-batch which contains a few data samples [44].

This sequential process creates many challenges in parallelizing the training. The current parallelism schemes are introduced below.

***Data Parallelism***

In data parallelism (Figure 3-1), every worker of the cluster hosts a full copy of the whole DNN model. The whole dataset is divided into non-overlapping partitions and each worker is assigned one partition. Multiple workers can concurrently train the model with different partitions. During the training process, the parameters needs to be exchanged and synchronized between workers through centralized or decentralized methods which will be introduced in the following sections.

The distributed algorithm for tuning the parameters contains two types, i.e. Synchronous Stochastic Gradient Descent (Sync-SGD) [51] and Asynchronous Stochastic Gradient Descent (Async-SGD) [52]. Sync-SGD consists of multiple training rounds. In each round, workers calculate the gradients using their local mini-batch; gradients of all workers need to be aggregated through a central server or peer-to-peer based approach, and finally the newly updated parameters will be sent to all workers which will start the next round. Sync-SGD suffers from the straggler effect, i.e. one slow worker may delay the whole training process. Async-SGD is not round based. Each worker obtains the latest parameters from a central parameter server, calculates gradients using a local mini-batch of data, and sends out the results to the server which updates the parameters based on the newly arrived gradients. Async-SGD suffers from the stale gradient problem, i.e. one worker may overwrite the global parameters which have been updated by other workers.

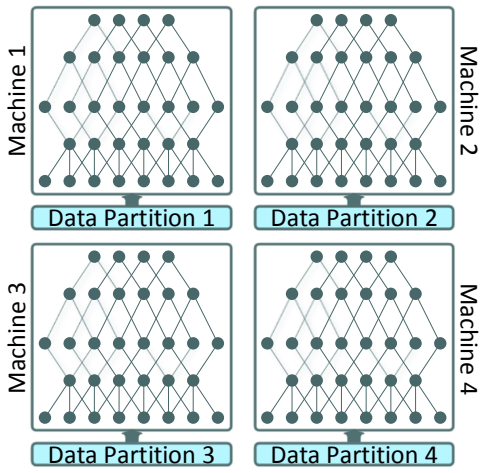



***Figure 3-1: Data Parallelism for Deep Learning***

***Model Parallelism***

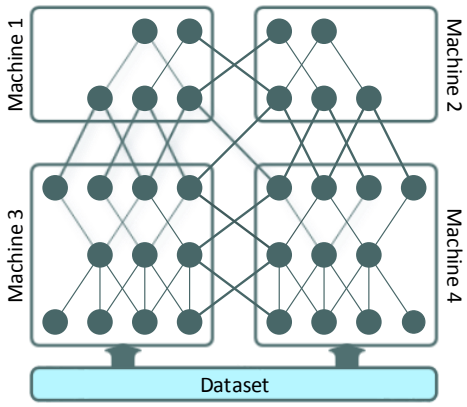
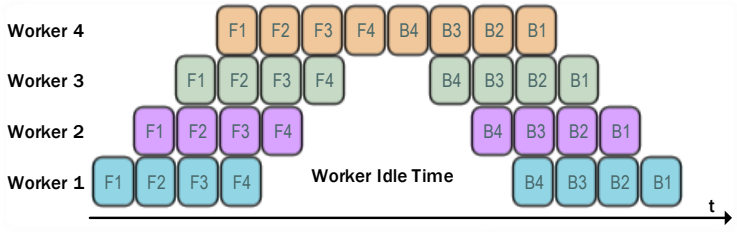



***Figure 3-2: Model Parallelism for Deep Learning***

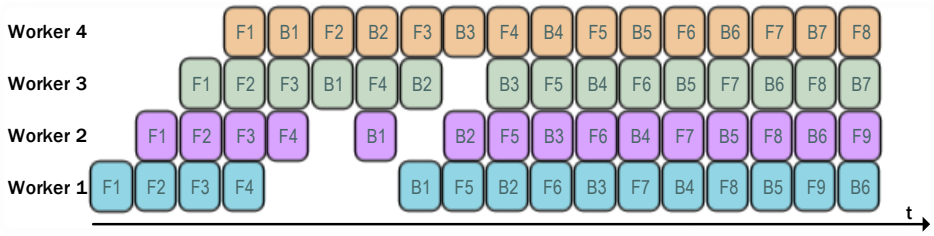
In model parallelism (Figure 3-2), the whole DNN model is partitioned and each worker maintains one partition. The input dataset is also logically divided according to the partition scheme of the input layer. This is different from that in data parallelism. In model parallelism, the dataset of the same mini-batch is split vertically and the partitions are fed into the worker with the correspondent input layer, whereas in data parallelism, the dataset is split horizontally and each worker processes data of different mini-batches. In the forward and backward propagation process, data pass through workers according to the order of layers.

Model parallelism provides a solution when the model is too large to fit in the memory of a single machine. However, a major challenge in minimizing the training time is to design the model partition scheme according to the capabilities of the underlying infrastructure including computational, storage and networking capabilities, which is a NP-Complete problem [53]. The sections below will introduce more details on the model partitioning approaches in hybrid edge and cloud environments.

***Pipeline Parallelism***




(a) Gpipe [49]



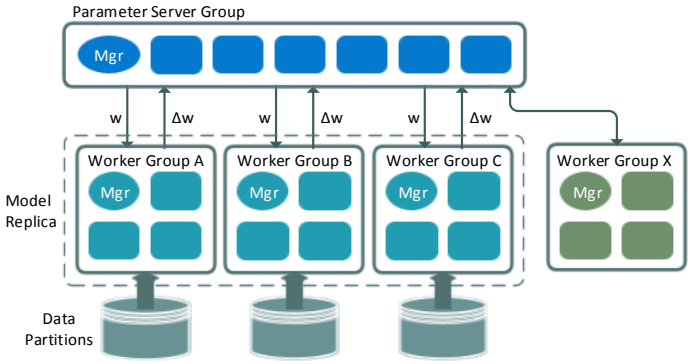

(b) Pipedream [50]

***Figure 3-3: Pipeline Parallelisms***

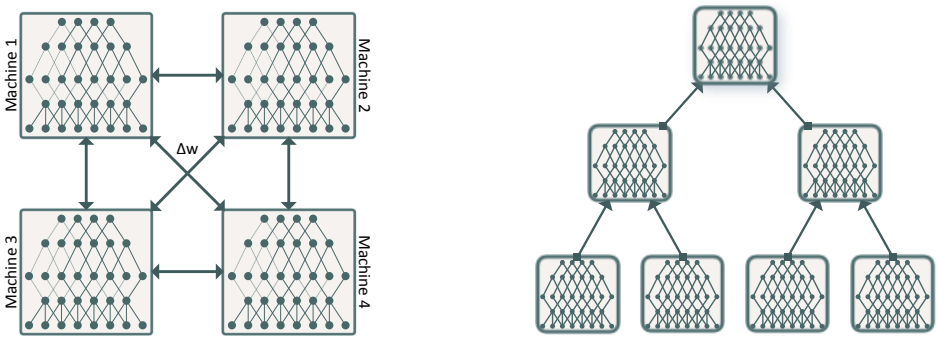
In model parallelism, each worker starts to work only after the results from the previous workers have been received. Therefore, only one worker is working at any point of time. Pipeline parallelism (Figure 3-3a) is proposed to resolve the problem. Gpipe [49] proposes to cut a mini-batch into multiple micro-batches. When a worker has transmitted a processed micro-batch to the next worker, both workers can work simultaneously. A smaller micro-batch size leads to a shorter idle time as shown in Figure 3-3a. In the backward propagation stage, the parameter updates need special care to guarantee correctness of the gradient calculation. Gpipe uses the same parameters for all micro-batches within a mini-batch. Parameters are updated at the end of each mini-batch. Pipedream [50] (Figure 3-3b) feeds mini-batches consecutively into the pipeline as Gpipe does for micro-batches. However, Pipedream takes a different scheduling approach. Each worker interleaves the processing of forward and backward propagation so that no workers are idle in the steady state. To ensure the correctness of the training, active parameters in the model are saved into different versions. The same set of parameters is applied to the same forward and backward propagation process.

### 3.2.2 Parameter Exchange Architectures

Parameter exchange between workers in model parallelism is straightforward, whereas in data parallelism, parameter synchronization between workers is a major challenge. Stochastic gradient descent is the current dominant algorithm for training deep neural networks. During the training, workers need to access shared parameters to refine the model. Different distributed computation architectures have been proposed to implement the data parallelism as shown in Figure 3-4.




(a) Parameter Server




(b) Peer-to-Peer

(c) AllReduce

***Figure 3-4: Parameter Server, Peer-to-Peer and AllReduce Architecture***

The parameter server architecture [43] [45] consists of one server group and a number of worker groups with one manager in each group to coordinate the server or worker nodes. The architecture supports running of multiple machine learning algorithms simultaneously. Parameter namespaces are used to isolate different algorithms. The parameters of each namespace is partitioned and duplicated across servers to achieve reliability and scalability. Multiple worker groups can share the same

namespace to query or update global parameters. The parameter server architecture is defined for data parallelism. However, it can be extended to support model parallelism by splitting models across workers as shown in Figure 3-4a. During the training process, each worker node runs the whole model on a data partition and sends parameter updates (subgradients) to the parameter server group. The server group calculates a new set of parameters based on the subgradients and sends new parameters to the workers.

In the peer-to-peer based architecture (Figure 3-4b), each peer sends parameter updates to all other peers directly. This scheme can remove the communication bottleneck between workers and parameter servers but also increase traffic between workers. Therefore, parameter compression and reducing the parameter updating rates are proposed to reduce the traffic [54].

Both parameter server and the peer-to-peer architectures can be enhanced by employing the AllReduce operation (Figure 3-4c) to reduce traffic, i.e. aggregate data through a reduce tree [55] [56]. AllReduce is a primitive of Message Passing Interface (MPI) which is widely used in parallel computing. It logically contains a Reduce and a Broadcast phase, i.e. a dataset is reduced to a smaller set by functions and then the result is broadcasted to all workers.

## 3.3 Deep Learning Technologies for the Edge

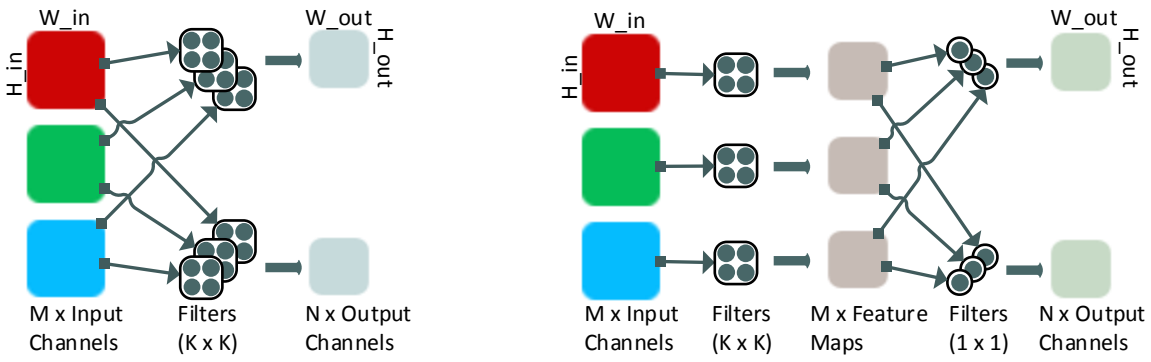
### 3.3.1 Lightweight Models for the Edge

***Group convolution***

Group convolution is a common technique for reducing the CNN model parameters, which was first proposed in AlexNet [57] with the aim of splitting the model into two GPUs.

A convolutional layer is to transform a W_in x H_in x M dimension feature map into a W_out x H_out x N dimension feature map (Figure 3-5a), where W_in, H_in, W_out and H_out are the spatial width and height of the input and output feature maps, and M and N are the depth of input and output feature maps (i.e. the number of channels). In the standard CNNs, each point on a W_out x H_out output channel is calculated by the sum of the results after applying one K x K dimension filter (kernel) on a certain region of each input channel. Therefore, the total computation steps of the standard convolution layer is K x K x M x N x W_out x H_out.

Group convolution separates input channels and filters into groups so that each group of filters are only applied to the correspondent input channels of the same group (instead of all input channels in the standard convolution). Consequently, computation costs are reduced. An extreme version of group convolution is to set each input channel as a group, which becomes depthwise separable convolution as described in Xception [58].




(a) Standard CNN (b) MobileNet

***Figure 3-5: MobileNets Architecture***

Mobilenet [59] proposes a lightweight CNN architecture for resource constraint devices using depthwise separable convolution. In order to reduce the parameter size, MobileNet splits the standard convolution into two steps, i.e. depthwise convolution (filtering) and pointwise convolution (combining), as illustrated in (Figure 3-5b). In the first step, each input channel is applied with one K x K filter, and consequently M input channels generate M intermediate feature maps. The second step is similar to the standard convolution except the filter size is 1 x 1. The M intermediate feature maps are combined together N times with different sets of weights to generate N output maps which will be the input of the next layer. The computation steps of MobileNet is K x K x W_out x H_out x M + 1 x 1 x M x N x W_out x H_out. The ratio of the computation cost of MobileNet to the standard CNN is 1/N+1/(KxK). MobileNet can achieves 8 to 9 times less computation costs than standard CNN while using 3x3 depthwise separable convolutions [59]. The performance of MobileNet is further improved by MobileNetV2 [60] through introducing an inverted residual structure with a linear bottleneck.

As the pointwise convolution is performed over all input feature maps, the computation cost can be further reduced by applying group convolution. However, if two consecutive layers employ group convolutions, a side effect is that each output channel is only generated from a small portion of input channels. ShuffleNet [61] proposes a shuffle operation between two group convolution layers to remove the side effect.

***Other Approaches in Reducing CNN Parameters***

Besides group convolution, various approaches to reduce CNN parameters have been proposed. SqueezeNet [62] reduces parameters by three strategies: 1) Replacing majority of 3x3 filters with 1x1 filters; 2) Reducing the input channel numbers for the 3x3 filters; 3) Delaying down sampling layers (pooling). Based on these design strategies, SqueezeNet proposes a Fire module which consists of one squeeze convolution layer and one expand layer. The squeeze convolution layer only contains 1x1 filters, whereas the expand layer contains a mixture of 1x1 and 3x3 filters. SqueezeNet can achieve AlexNet level accuracy on ImageNet with 50 times less parameters. Deep Fried Convnets [63] proposes to reduce the parameters in the CNN fully connected layer through reparameterizing the matrix-vector multiplication using Adaptive Fastfood transform.

***Network Compression***

Q-CNN [64] proposes to reduce memory footprint and increase computation efficiency through quantization of the parameters. Xnor-net [65] proposes to approximate the weights and the intermediate inputs to the convolution layers and fully connected layers through binary values. HashNet [66] employs a hash function to group weights and the same group share the same weight. DeepCompression [67] proposes to compress the model through a 3-stage pipeline, i.e. pruning, quantization and Huffman coding. Knowledge distillation provides a network compression approach by transferring knowledge from a larger teacher model to a smaller student network [68] [69] [70].

### 3.3.2 Federated Learning

The datasets generated in local devices, e.g. mobile phones and CCTV cameras, may contain privacy sensitive information. Current cloud based training methods require to upload the dataset from local devices to the cloud, which may violate users' privacy. Federated learning denotes a distributed learning technique which trains a shared model from distributed dataset which remains in the local devices. In [71], the authors propose a federated learning approach which is similar to data parallelism, i.e. each device computes parameter updates and sends the gradients to a server which performs FederatedAveraging. A specific number of devices are randomly selected in each round to calculate parameter updates. The chosen devices use the predefined mini-batch size and epoch number to calculate the parameters and send the results to the server for aggregation. In [72], the authors propose two ways to reduce communication costs while uploading parameters to the server, i.e. structured updates and sketched updates. In [73], the authors propose to increase the learning accuracy by allowing a small portion of globally shared datasets.

### 3.3.3 Improving Communication Efficiency in Data Parallelism

***Gradient Quantisation***

Communication costs is a well-known bottleneck in distributed DNN training while exchanging gradients and parameters between nodes. In [74], the authors propose a sub-gradient quantisation scheme to reduce the communication costs for data parallelism based DNN training using Sync-SGD. The gradients can be represented by 1 bit. The key concept is to keep the quantisation error and add the error to the respective gradient of the following mini-batch before quantisation. Terngrad [75] employs ternary gradients, i.e. {-1, 0, 1}, for accelerate distributed DNN training with data parallelism and Sync-SGD; it employs layer-wise ternarizing and gradient clipping to improve convergence. QSGD [76] provides a gradient quantisation mechanism to balance the communication cost and convergence time. The bits to be communicated can be specified per iteration.

***Gradient Sparsification***

During data parallelism based training, the calculated sub-gradients are normally sparse and consequently the communication costs can be reduced by only sending the significant gradients to the parameter server or other nodes. This approach is called gradient sparsification. A key challenge is to decide which gradients are significant to the parameter updating process. ESGD [77] compares the current and previous values of the loss function. If the loss becomes smaller, the algorithm will continue using the coordinates of the currently used gradients. Otherwise, the gradients will be randomly selected based on the weight value. DGC (Deep Gradient Compression) [78] only transmits gradients above a threshold but keeps other gradients locally which will be transmitted when the accumulated value is above the threshold. In [79], the authors propose to minimize the coding length of gradients using convex optimisation. The gradients are dropped out based on certain probabilities and the reserved gradients are amplified according to the correspondent probability values to remove the bias introduced by sparsification.

### 3.3.4 Model Parallelism based Distributed Inference

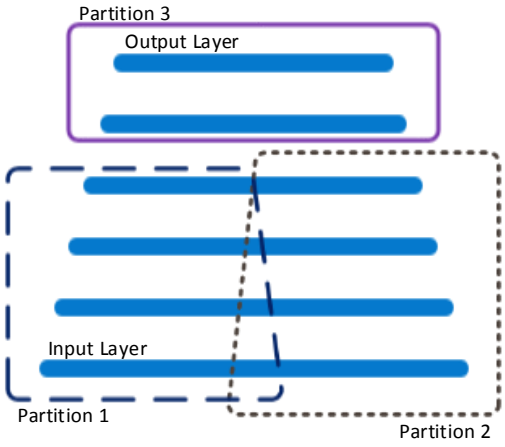



***Figure 3-6: DeepThings Model Partition Scheme [81]***

As the footprint of a full DNN model may exceed the capacity of an edge device, one solution is to spread the trained model across multiple edge devices. MoDNN [80] proposes a MapReduce based model to partition a pre-trained DNN model into a number of mobile devices. DeepThings [81] proposes a partition scheme for deploying early stage layers of CNN in multiple resource constrained IoT edge devices. CNN behaves like a compressor which extracts visual features and removes redundant information from the images. Therefore, early stage layers generate much larger dimensional data than later stage layers. Consequently, they require much higher memory footprints and communication costs (if transmitted) than later stage layers. As a result, a resource constrained device may not be able to host even one single early stage layer. As each neuron in a CNN layer only

depends on a few local neurons in its previous layers, CNN layers can be partitioned vertically to include all required neurons of each layer for calculating the top layer neurons in the same partition (as shown in Figure 3-6). These partitions can be executed independently by individual edge devices without interacting with other vertical partitions. The output of these partitions need to be aggregated to generate the final inference result. This horizontal partition can be performed over an intermediate layer with a low data output.

### 3.3.5 DNN Models with Early Exits

In edge environments, IoT devices have limited computational resources. Deploying a shallow neural network in these devices may suffer from low inference accuracies, whereas deploying a large neural network model in the cloud may suffer from long latencies. Using model parallelism to split the model between IoT devices and the cloud may not enhance the inference performance due to long network delays. DDNN [82] and Branchynet 0 propose a distributed and hierarchical deep neural network architecture with the layers deployed across IoT devices, edge cloud and central cloud. DDNN is different from model parallelism is that DDNN introduces local exits at certain point along the inference path, e.g. at the device and/or edge levels. The design rationale is that if inference results have reached a certain accuracy level, there is no need to pass current results to upper layers. Consequently, communication costs can be significantly reduced.

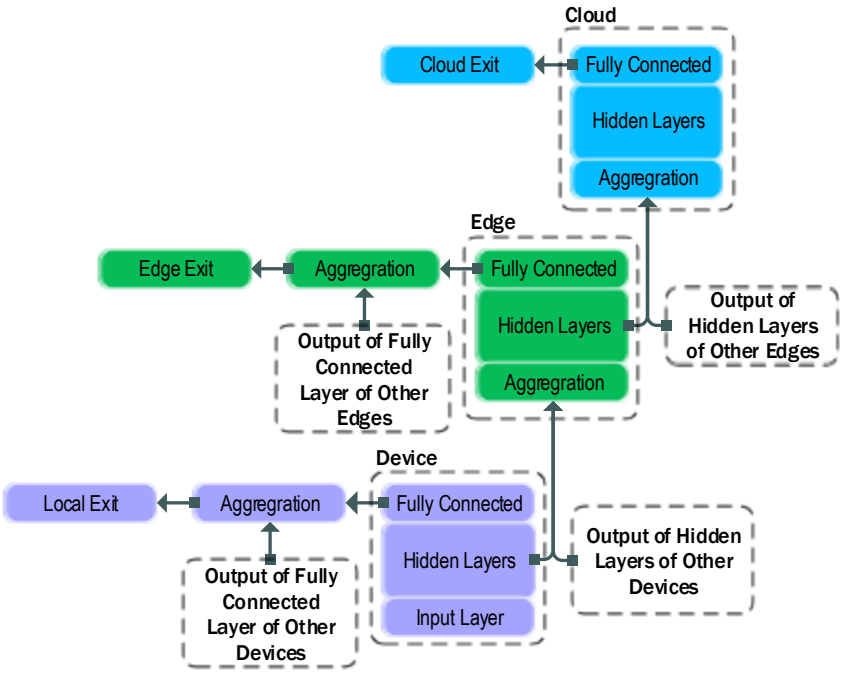



***Figure 3-7: DDNN Architecture***

The DDNN architecture is shown in Figure 3-7. Each IoT device receives input data and processes it through a full shallow neural network. The inference results from a group of devices are aggregated through a local function, e.g. max pooling, average pooling or concatenation. A normalized entropy threshold is used as the inference confidence criteria. If the confidence is above a certain threshold, the inference for the current input completes. Otherwise, intermediate inference results are sent to the higher level aggregation function which prepares input data for upper layers. This process continues until the inference reaches the final layer. Note that intermediate inference results that are sent to the upper layer is the same as the input data to the fully connected layer at the local level.

# 4 Summary

Both edge computing and distributed artificial intelligence are fast growing research areas, and are tightly coupled with many other technological areas, e.g. Internet of Things, Big Data, 5G, Cloud

Computing, Smart Manufacturing, and Autonomous Vehicles. This chapter focuses on a specific perspective of these areas, i.e. the challenges in deploying, executing, and managing distributed intelligent applications over the edge to cloud continuum. More specifically, this chapter has discussed the background of the emergence of edge computing, the approaches for managing heterogeneous computational resources over the edge and cloud, the benefits, challenges and uses cases of deploying distributed artificial intelligence and data analytics applications across edge and cloud, and various distributed deep learning technologies that can potentially be employed in edge applications. Besides the above mentioned technologies, there are many other challenges in these areas, e.g. managing the data from things and intermediate devices, and creating a complete edge to cloud computing ecosystem encompassing heterogeneous networking, storage, and computational resources from different providers. Furthermore, current distributed deep learning algorithms for edge computing are mainly designed for inference. Training at the edge is still at the infant stage.

## References


[1] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog Computing for the Internet of Things : A Survey," vol. 19, no. 2, 2019.

[2] A. Yousefpour et al., "All one needs to know about fog computing and related edge computing paradigms: A complete survey," Journal of Systems Architecture. 2019.

[3] C.-H. Hong and B. Varghese, "Resource Management in Fog/Edge Computing: A Survey," 2018.

[4] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, "Security and Privacy for Cloud-Based IoT: Challenges," IEEE Commun. Mag., vol. 55, no. 1, pp. 26–33, 2017.

[5] J. Pascoe, N. Ryan, and D. Morse, Handheld and ubiquitous computing : First International Symposium, HUC '99, Karlsruhe, Germany, September 27-29, 1999 : proceedings. 1999.

[6] R. K. Naha et al., "Fog computing: Survey of trends, architectures, requirements, and research directions," IEEE Access, vol. 6, pp. 47980–48009, 2018.

[7] S. Verma, et al., "A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues," in IEEE Comm. Surveys & Tutorials, vol. 19, no. 3, pp. 1457-1477, 2017.

[8] P. Wood, et al."Dependability in Edge Computing," CoRR abs/1710.11222, 2017.

[9] R. Yu, G. Xue, and X. Zhang, Application provisioning in fog computing-enabled internet-of-things: A network perspective, IEEE INFOCOM 2018, pp. 783-791, 2018

[10] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, Resource provisioning for iot services in the fog, IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 32-39, 2016

[11] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, Optimized iot service placement in the fog, Service Oriented Computing and Applications, vol. 11, pp. 427-443, 2017

[12] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, Cost efficient resource management in fog computing supported medical cyber-physical system, IEEE Transactions on Emerging Topics in Computing, vol. 5, pp. 108-119, 2017

[13] Y. Cao, S. Chen, P. Hou and D. Brown, FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation, IEEE NAS, pp. 2-11, 2015

[14] C. Perera, D. S. Talagala, C. H. Liu, and J. C. Estrella, Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in IoT clouds, IEEE Transactions on Computational Social Systems, vol. 2, pp. 171–181, 2015

[15] T. N. Gia, M. Jiang, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, Fog computing in healthcare internet of things: A case study on ECG feature extraction, IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 356-363, 2015

[16] G. López, V. Custodio, and J. I. Moreno, Lobin: E-textile and wireless sensor- network-based platform for healthcare monitoring in future hospital environments, IEEE Transactions on Information Technology in Biomedicine, vol. 14, pp. 1446-1458, 2010
[17] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan and G. J. Ren, Foggy clouds and cloudy fogs: A real need for coordinated management of fog-to-cloud computing systems, IEEE Wireless Comm., vol. 23, pp. 120-128, 2016
[18] A. Benoit, H. Casanova, V. Rehn-Sonigo and Y. Robert, Resource Allocation for Multiple Concurrent In-network Stream-Processing Applications, European Conference on Parallel Processing, pp 81-90, 2009
[19] Z. Shen, V. Kumaran, M.J. Franklin, S. Krishnamurthy, A. Bhat, M. Kumar, R. Lerche and K. Macpherson, CSA: streaming engine for internet of things. IEEE Data Eng. Bull. 38 (4), pp. 39–50, 2015
[20] R. Eidenbenz and T. Locher, Task allocation for distributed stream processing, IEEE INFOCOM, pp. 1-9, 2016
[21] J. Huang, G. Liu, Q. Duan, and Y. Yan, Qos-aware service composition for converged network-cloud service provisioning, IEEE International Conference on Services Computing, pp. 67–74, 2014
[22] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, Optimal operator placement for distributed stream processing applications, ACM International Conference on Distributed and Event-based Systems, pp. 69–80, 2016
[23] R. Mayer, L. Graser, H. Gupta, and E. Saurez, EmuFog: Extensible and Scalable Emulation of Large-Scale Fog Computing Infrastructures, IEEE Fog World Congress, pp. 1-6, 2017
[24] Brian Ray, “The Role of Cloud Computing and Fog Computing in IoT”, online 2017: “https://www.iotforall.com/cloud-fog-computing-iot/
[25] Quang Duy La, Mao V. Ngo, Thinh Quang Dinh, Tony Q.S. Quek, Hyundong Shin, Enabling intelligence in fog computing to achieve energy and latency reduction, Digital Communications and Networks, Volume 5, Issue 1, 2019,
[26] Zane Tsai, “The Emerging Role of AI in Edge Computing”, online 2019: ttps://www.rtinsights.com/the-emerging-role-of-ai-in-edge-computing/
[27] David Schatsky, Navya Kumar and Sourabh Bumb, Deloite Insights, online 2017: https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/intelligent-iot-internet-of-things-artificial-intelligence.html
[28] i-SCOOP: Edge computing and IoT 2018 – when intelligence moves to the edge, online: https://www.i-scoop.eu/internet-of-things-guide/edge-computing-iot/
[29] Kraska, Tim et al. “MLbase: A Distributed Machine-learning System.” CIDR (2013). Online: http://cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf
[30] Elad Hazan and Tomer Koren. The computational power of optimization in online learning. In Proc. 48th Symposium on Theory of Computing, pages 128–141, 2016.
[31] O. B. Sezer, E. Dogdu and A. M. Ozbayoglu, "Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey," in IEEE Internet of Things Journal, vol. 5, no. 1, pp. 1-27, Feb. 2018. doi: 10.1109/JIOT.2017.2773600
[32] The interaction and convergence of IoT and AI at work Online: https://www.i-scoop.eu/internet-of-things-guide/iot-and-ai/
[33] Mitsubishi Electric, e-Factory, November 2016 [Online]. Available: http://app.mitsubishielectric.com/app/fa/download/search.do?kisyu=/sol/efactory&mode=catalog. [Accessed 08 May 2019].
[34] Mohanty S., Vyas S. (2018) Intelligence of Things = IoT + Cloud + AI. In: How to Compete in the Age of Artificial Intelligence. Apress, Berkeley, CA
[35] White Paper, Niagara Framework, Niagara Appliance, online 2018: https://pages1.tridium.com/NiagaraIIoT.html

[36] M. Mohammadi, A. Al-Fuqaha, M. Guizani and J. Oh, "Semi-supervised Deep Reinforcement Learning in Support of IoT and Smart City Services," in IEEE Internet of Things Journal, vol. 5, no. 2, pp. 624-635, April 2018.doi:10.1109/JIOT.2017.2712560
[37] Agriculture goes Digital, online: https://www.schneider-electric.com/en/work/campaign/life-is-on/case-study/waterforce.jsp
[38] LeCun, Yann. "Generalization and network design strategies." Connectionism in perspective (1989): 143-155.
[39] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." Neural computation 18, no. 7 (2006): 1527-1554.
[40] Salakhutdinov, Ruslan, and Geoffrey Hinton. "Deep boltzmann machines." In Artificial Intelligence and Statistics, pp. 448-455. 2009.
[41] Martens, James, and Ilya Sutskever. "Learning recurrent neural networks with hessian-free optimization." In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1033-1040. 2011.
[42] Fadlullah, Zubair, Fengxiao Tang, Bomin Mao, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems." IEEE Communications Surveys & Tutorials (2017).
[43] Dean, Jeffrey, et al. Large scale distributed deep networks. In: Advances in neural information processing systems. 2012. p. 1223-1231.
[44] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT Press, 2016.
[45] Li, Mu, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. "Scaling Distributed Machine Learning with the Parameter Server." In OSDI, vol. 14, pp. 583-598. 2014.
[46] Raina, Rajat, Anand Madhavan, and Andrew Y. Ng. "Large-scale deep unsupervised learning using graphics processors." In Proceedings of the 26th annual international conference on machine learning, pp. 873-880. ACM, 2009.
[47] Agarwal, Alekh, Olivier Chapelle, Miroslav Dudík, and John Langford. "A reliable effective terascale linear learning system." Journal of Machine Learning Research 15, no. 1 (2014): 1111-1133.
[48] Mayer, R. and Jacobsen, H.A., 2019. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools. arXiv preprint arXiv:1903.11314.
[49] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q.V. and Chen, Z., 2018. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1811.06965.
[50] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G. and Gibbons, P., 2018. Pipedream: Fast and efficient pipeline parallel dnn training. arXiv preprint arXiv:1806.03377.
[51] Chen, J., Pan, X., Monga, R., Bengio, S. and Jozefowicz, R., 2016. Revisiting distributed synchronous SGD. arXiv preprint arXiv:1604.00981.
[52] Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.M. and Liu, T.Y., 2017, August. Asynchronous stochastic gradient descent with delay compensation. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 4120-4129). JMLR. org.
[53] Mayer, R., Mayer, C. and Laich, L., 2017, December. The tensorflow partitioning and scheduling problem: it's the critical path!. In Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning (pp. 1-6). ACM.
[54] Strom, Nikko. "Scalable distributed DNN training using commodity GPU cloud computing." In INTERSPEECH, vol. 7, p. 10. 2015.
[55] Iandola, Forrest N., Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. "FireCaffe: near-linear acceleration of deep neural network training on compute clusters." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2592-2600. 2016.

[56] Mai, Luo, Chuntao Hong, and Paolo Costa. "Optimizing Network Performance in Distributed Machine Learning." In HotCloud. 2015.
[57] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
[58] Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).
[59] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
[60] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).
[61] Zhang, X., Zhou, X., Lin, M. and Sun, J., 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6848-6856).
[62] Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360.
[63] Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L. and Wang, Z., 2015. Deep fried convnets. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1476-1483).
[64] Wu, J., Leng, C., Wang, Y., Hu, Q. and Cheng, J., 2016. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4820-4828).
[65] Rastegari, M., Ordonez, V., Redmon, J. and Farhadi, A., 2016, October. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision (pp. 525-542). Springer, Cham.
[66] Chen, W., Wilson, J., Tyree, S., Weinberger, K. and Chen, Y., 2015, June. Compressing neural networks with the hashing trick. In International Conference on Machine Learning (pp. 2285-2294).
[67] Han, S., Mao, H. and Dally, W.J., 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.
[68] Hinton, G., Vinyals, O. and Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
[69] Kim, Y. and Rush, A.M., 2016. Sequence-level knowledge distillation. arXiv preprint arXiv:1606.07947.
[70] Huang, Z. and Wang, N., 2017. Like what you like: Knowledge distill via neuron selectivity transfer. arXiv preprint arXiv:1707.01219.
[71] McMahan, H.B., Moore, E., Ramage, D. and Hampson, S., 2016. Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629.
[72] Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T. and Bacon, D., 2016. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492.
[73] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D. and Chandra, V., 2018. Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.
[74] Seide, F., Fu, H., Droppo, J., Li, G. and Yu, D., 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In Fifteenth Annual Conference of the International Speech Communication Association.
[75] Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y. and Li, H., 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In Advances in neural information processing systems (pp. 1509-1519).

[76] Alistarh, D., Grubic, D., Li, J., Tomioka, R. and Vojnovic, M., 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In Advances in Neural Information Processing Systems (pp. 1709-1720).

[77] Tao, Z. and Li, Q., 2018. esgd: Communication efficient distributed deep learning on the edge. In {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18).

[78] Lin, Y., Han, S., Mao, H., Wang, Y. and Dally, W.J., 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887.

[79] Wangni, J., Wang, J., Liu, J. and Zhang, T., 2018. Gradient sparsification for communication-efficient distributed optimization. In Advances in Neural Information Processing Systems (pp. 1299-1309).

[80] Mao, J., Chen, X., Nixon, K.W., Krieger, C. and Chen, Y., 2017, March. Modnn: Local distributed mobile computing system for deep neural network. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017 (pp. 1396-1401). IEEE.

[81] Zhao, Z., Barijough, K.M. and 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), pp.2348-2359.

[82] Teerapittayanon, S., McDanel, B. and Kung, H.T., 2017, June. Distributed deep neural networks over the cloud, the edge and end devices. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (pp. 328-339). IEEE.

Teerapittayanon, S., McDanel, B. and Kung, H.T., 2016, December. Branchynet: Fast inference via early exiting from deep neural networks. In 2016 23rd International Conference on Pattern Recognition (ICPR) (pp. 2464-2469). IEEE.