Enabling Deep Learning for All-in EDGE paradigm

Praveen Joshi¹, Haithem Afli¹, Mohammed Hasanuzzaman¹, Chandra Thapa², and Ted Scully¹
¹Munster Technological University, Rossa Ave, Bishopstown, Cork, T12 P928, Ireland
²CSIRO Data61, Sydney, Australia

ABSTRACT
Deep Learning-based models have been widely investigated, and they have demonstrated significant performance on non-trivial tasks such as speech recognition, image processing, and natural language understanding. However, this is at the cost of substantial data requirements. Considering the widespread proliferation of edge devices (e.g., Internet of Things devices) over the last decade, Deep Learning in the edge paradigm, such as device-cloud integrated platforms, is required to leverage its superior performance. Moreover, it is suitable from the data requirements perspective in the edge paradigm because the proliferation of edge devices has resulted in an explosion in the volume of generated and collected data. However, there are difficulties due to other requirements such as high computation, high latency, and high bandwidth caused by Deep Learning applications in real-world scenarios. In this regard, this survey paper investigates Deep Learning at the edge, its architecture, enabling technologies, and model adaption techniques, where edge servers and edge devices participate in deep learning training and inference. For simplicity, we call this paradigm the All-in EDGE paradigm. Besides, this paper presents the key performance metrics for Deep Learning at the All-in EDGE paradigm to evaluate various deep learning techniques and choose a suitable design. Moreover, various open challenges arising from the deployment of Deep Learning at the All-in EDGE paradigm are identified and discussed.

KEYWORDS
Artificial intelligence, All-in EDGE paradigm, deep learning, distributed systems, decentralized systems

1 INTRODUCTION
The global community is increasingly becoming a data-driven environment, in which systems are generating vast quantities of data outside of the traditional data centers. Cisco anticipates that global Internet traffic will be 396 Exabytes (EB) per month in 2022, up from 122 EB per month in 2017 [43]. This enormous amount of data has a positive impact on Artificial Intelligence (AI) applications. In particular, Deep Learning (DL) techniques rely on the availability of large quantities of data [4, 153].

In recent years, DL has shown promising progress in natural language processing, computer vision, and big data analysis. Examples include natural language processing tasks where models like BERT, Megatron-LM, GPT3, Grapher, etc. [44, 50, 85, 201] are reaching human-level understanding of the textual data. Other examples include DL techniques that have exceeded human performance on object classification tasks such as ImageNet [16, 117] or have outperformed humans in the game of Multiplayer Online Battle Arena without any human intervention [276].

DL requires significant compute resources to facilitate both training and inference phase; this is mainly achieved by utilizing Cloud or in-house computing infrastructures. The need to collect, process, and transfer the vast quantity of data to the central Cloud can become a bottleneck in many mission-critical use-cases [205, 236]. However, EDGE computing acts as a high-performance bridge from local systems to private and public Clouds. EDGE computing, which typically has relatively small hardware and memory footprint, can provide a valuable infrastructure at the network’s periphery. EDGE computing has typically performed tasks such as collection, filtering, and light-weight computation of raw data. Consequently, the provision of such processing at the EDGE means that only the necessary processed data needs to be transferred to the centralized data center instead of the transmission of all raw data in earlier days [8].

In addition, with progress in Deep Learning-based architectures and algorithms, training and inference are now being pushed to the network’s EDGE.

The convergence of AI and EDGE computing has given rise to a new paradigm of research, called EDGE Intelligence (EI) or EDGE AI [79, 272] with the goal of facilitating AI modelling closer to the data generation source. In the last couple of years, there has been a significant increase in the number of research papers published in the domain of EDGE Intelligence. In the area of Edge intelligence there has been a 150% increase in published papers between 2016 and 2021. Figure 1 depicts the interest of academic researchers in the field of EDGE intelligence [51].

EDGE Intelligence enables empowering new applications and innovations that leverage its proximity to the end-user [145]. These opportunities have been recognized by both industry and academia. Companies such as Google, IBM, and Microsoft have developed more powerful EDGE servers [26] while DL at the EDGE is being used across various application domains, including video analytics [207], healthcare [9], natural language processing [258], network functions [279] and virtual and augmented reality [150].

Figure 1: Publication volume over time for the topic EDGE Intelligence.
EI is more focused on decentralized and distributed architectures in order to utilize EDGE servers, which are available in high numbers and close to the point of data generation. Therefore, while an individual server may have relatively low computing power for facilitating DL training and inference, a collection of EDGE servers can be leveraged to effectively train and infer from DL techniques at the EDGE. A few prominent examples are Distributed Machine Learning-based technologies [314], Federated Learning [2] and Split Learning [73], which are explained in more detail in this paper in section 3.2. Also, to infer from the DL model at EDGE from the resources-constrained EDGE server, it is required most of the time to adapt the DL model by compressing the model’s size or by using the conditional computation as explained in section 3.3. This is to adapt the DL model by compressing the model’s size or by applying the conditional computation of the DL training and inference.

The adoption of EI also helps to mitigate a range of specific challenges inherent in the traditional EDGE-Cloud architecture. These include addressing data privacy issues, challenges around DL problems which arise when DL is pushed to the All-in EDGE paradigm. This survey looks upon the underneath mentioned EDGE DL computing paradigm [6, 11], which is the foundation for this survey. To the best of our knowledge, none of the mentioned surveys looked upon the enablers and challenges from the All-in EDGE paradigm perspective. All in all, considering the aforementioned questions, the contributions of this survey paper are the following:

(1) How do we formally define the emergent All-in EDGE paradigm?
(2) How do different architectures (centralised, decentralised and distributed) work in the All-in EDGE paradigm?

EI is more focused on decentralized and distributed architectures in order to utilize EDGE servers, which are available in high numbers and close to the point of data generation. Therefore, while an individual server may have relatively low computing power for facilitating DL training and inference, a collection of EDGE servers can be leveraged to effectively train and infer from DL techniques at the EDGE. A few prominent examples are Distributed Machine Learning-based technologies [314], Federated Learning [2] and Split Learning [73], which are explained in more detail in this paper in section 3.2. Also, to infer from the DL model at EDGE from the resources-constrained EDGE server, it is required most of the time to adapt the DL model by compressing the model’s size or by using the conditional computation as explained in section 3.3. This is to adapt the DL model by compressing the model’s size or by applying the conditional computation of the DL training and inference.

The adoption of EI also helps to mitigate a range of specific challenges inherent in the traditional EDGE-Cloud architecture. These include addressing data privacy issues, challenges around DL problems which arise when DL is pushed to the All-in EDGE paradigm. This survey looks upon the underneath mentioned EDGE DL computing paradigm [6, 11], which is the foundation for this survey. To the best of our knowledge, none of the mentioned surveys looked upon the enablers and challenges from the All-in EDGE paradigm perspective. All in all, considering the aforementioned questions, the contributions of this survey paper are the following:

(1) How do we formally define the emergent All-in EDGE paradigm?
(2) How do different architectures (centralised, decentralised and distributed) work in the All-in EDGE paradigm?

(3) What is the state of the art in training (and inference) enablers for the All-in EDGE paradigm?
(4) What are the standard Key Performance metric to compute for the All-in EDGE paradigm?

Unlike prior surveys [30, 46, 178, 192, 271, 314] summarised in the Table 1, this survey focuses on providing insights into enablers for DL in the All-in EDGE paradigm. To the best of our knowledge, none of the mentioned surveys looked upon the enablers and challenges from the All-in EDGE paradigm perspective. All in all, considering the aforementioned questions, the contributions of this survey paper are the following:

(1) The survey provides clear distinctions in the computing paradigm and EDGE intelligence. This is required to find the networking infrastructure that can be exploited at different computing paradigms as well as different levels of EI.
(2) A state of the art review of enablers required for deep learning training and inference from All-in EDGE paradigm perspective.
(3) A discussion of the key performance metrics for the All-in EDGE paradigm. The metrics enable finding suitable methods for evaluating the deep learning model at the All-in EDGE paradigm.
(4) Identification of open challenges in the All-in EDGE paradigm from an operational perspective for the attention of the research community in academia and industry.

An overview of the survey paper organisation is provided in the Figure 2. The survey paper is organised as follows:

- Section II provides the primer on computing paradigm and EDGE intelligence (EI). This section also defines the all-in-EDGE paradigm of EI.
- Section III presents the architecture, enabling technologies for training and inference from the deep learning models at the All-in EDGE paradigm, and also examines model adaptation technique for effective deployment of deep learning models at EDGE.
- Section IV reviews the key performance metrics used for evaluating the research in all-in-EDGE deep learning.
- Section V discusses the open challenges and future direction of research in all-in-EDGE deep learning.
- Section VI presents a summary and identifies the primary conclusions and findings of the paper.

2 FUNDAMENTALS OF EDGE INTELLIGENCE AND ALL-IN EDGE LEVEL EI

The centralised nature of the Cloud data centre has certain drawbacks. One of the most considerable disadvantages is the distance between the data centres and end (user) devices. On the other hand, EDGE computing provides undeniable benefits in bringing storage and computational resources physically closer to the source of data generation, thereby reducing the AI models’ prediction latency. This section discusses the distinction between the Cloud and EDGE computing paradigms. This section also defines the different levels of EDGE intelligence based on the target (such as deep learning training and inference) performed by the discrete computing paradigms (Cloud and EDGE).
2.1 Computing paradigms
Facilitating deep learning can require significant computational and storage capacity. As such, this requirement can represent a substantive impediment to the deployment of deep learning models on IoT and other resource-constrained devices. Below we evaluate the Cloud and EDGE computing paradigms in the context of deep learning based on their storage, computational power, proximity to the ED.

2.1.1 Cloud Computing. Cloud servers have significant storage capacity and computational power to facilitate the overwhelming
data coming via the backhaul network from the end-user [38, 214]. Thus, Cloud data centres can satisfy resource requirements for aggregation, pre-processing, and inference for any AI-based applications.

The Cloud data centre resources are concentrated in a few data centres, providing global coverage with a backhaul network. The Cloud computing paradigm typically involves the end devices that offload data directly to the Cloud for further processing. The end devices mentioned here are the originators of the data. In the Cloud, data can persist for days, months, and years, meaning that long term data can be collated and processed. For example, Cloud data centres can facilitate forecasting models based on large amounts of historical time series data [199]. Cloud computing is still the appropriate vehicle for modelling and analytical processing if latency requirements and bandwidth consumption are not an issue, provided measures for preserving privacy and security are in place [45].

2.1.2 EDGE Computing. With the surge in the proliferation of IoT devices, traditional centralised Cloud computing struggles to provide an acceptable Quality of Service (QoS) level to the end customers [5]. EDGE computing coupled with advancements in networking technology and the advent of 5G technology represents a viable mechanism of resolving this issue [14, 116]. The EDGE computing paradigm has emerged due to the need to push computation away from the Cloud and towards the network’s EDGE. This allows us to place computing resources close to the data sources. EDGE devices (ED), which are inherent components of EDGE computing, are widely distributed across geographical regions, and as such they are in closer proximity to the origin of the data. Although being widely available for processing the intermittent time-sensitive data in the network, data in such EDGE devices are transient [180]. In contrast to Cloud computing, latency incurred from EDGE computing is significantly less, as a majority of data does not have to travel via a backhaul network to the Cloud [213]. Less consumption of backhaul networks also means the requirement of bandwidth consumption is considerably less. ED are located in closest proximity to the data source, as shown in the Figure 3.

2.2 EDGE intelligence

In recent years, there has been a significant amount of research in EDGE computing. EDGE computing can bring computing resources and storage capacity closer to the ED, thereby improving the QoS of real-time based applications such as automated driving [169] and real-time surveillance [17] all of which intrinsically require fast processing, and response time [82, 142]. Meanwhile, over the last decade, significant progress has been made in the AI domain. Technical advancements in high-performance processors [127] coupled with AI advancements in algorithms, and the availability and maturity of big data processing [216] have all contributed to the increase in AI performance. Applications ranging from Apple’s Siri and Microsoft’s Cortana to Alpha Go demonstrate AI technology’s pervasive and diverse impact over the last number of years. AI applications have established their candidacy as a necessary component of today’s data-driven world and its ability to extract meaningful information. Over the past number of years it has been widely recognised that the proliferation of EDGE devices and the subsequent vast amounts of generated data represents a high-potential application space for AI technology. As mentioned by the author in [314]: “EI is a paradigm that fully exploits the available data and resources across the hierarchy of end devices, EDGE nodes, and Cloud datacenters to optimise the overall performance of training and inferencing a DNN model.”

Considerable efforts have been invested in research exploring how EDGE devices can reduce the dependency on Cloud computing. For example, there has been significant research interest in minimising data transmission between EDGE devices and centralised Cloud servers as this can help significantly alleviate network congestion. EI can be divided into six levels based on participation and processing, as described in the Figure 4.

The first level involves participation from the Cloud and the EDGE servers. At this level, the Cloud is solely responsible for the training of the Deep Learning model. For the inference phase, a Cloud-EDGE based co-inference strategy is used, in which initially EDGE determines the output with a certain confidence. If the result produced by the model on EDGE server exceeds a fixed threshold probability then the result is provided to the application/end-user. If the probability threshold is not met then the activations of the last layer of the network on the EDGE server are passed to the Cloud for further processing.

The second level again involves the participation from the Cloud and EDGE servers. Like level 1, Cloud is responsible for training the Deep Learning model. Unlike level 1, the EDGE server is solely responsible for providing the predicted output. There are a range of techniques which will be discussed in more detail in section 3.2 that are used to condense deep learning models to facilitate deployment on EDGE servers.

Table 2: List of important abbreviations

| Abbreviation | Definition                        |
|--------------|----------------------------------|
| AI           | Artificial Intelligence          |
| EDGE         | Enhanced Data GSM Environment    |
| EI           | EDGE Intelligence                |
| DNN          | Deep Neural Network              |
| GDPR         | General Data Protection Regulation|
| CIPAA        | Construction Industry Payment and Adjudication Act |
| DL           | Deep Learning                    |
| ANN          | Artificial Neural Network        |
| DC           | Data Center                      |
| MEC          | Mobile (Multi-Access) EDGE Computing |
| CC           | Cloud Computing                  |
| EC           | EDGE Computing                   |
| IoT          | Internet of Things               |
| ED           | EDGE Devices                     |
| ES/N         | EDGE Server or EDGE Node         |
| ES           | EDGE Server                      |
The third level involves the participation of the Cloud and the end device. This level utilises the Cloud to train the deep learning model. The model is deployed and used for inference on the users’ end device.

The fourth level involves closer integration between the Cloud and EDGE servers. The fourth level is similar to the first except that more of the computational burden is pushed towards the EDGE. In contrast to level 1, in level 4, both the Cloud and EDGE servers are responsible for training the deep learning model. The inference phase still utilises a Cloud-EDGE based co-inference strategy.

The fifth level involves only EDGE servers. EDGE servers become solely responsible for training the deep learning model. Based on the deep learning model’s size, either a single EDGE server can train a deep learning model, or a group of EDGE servers collaborate to train a deep learning model. Techniques for training deep learning at level 5 are described in detail in section 3.2. Inference at level 5 can be produced from either a single EDGE server or multiple EDGE servers working collaboratively. Also, the fifth level of EI is referred to as the All-in-EDGE paradigm in this survey paper.

The sixth level involves only EDGE devices. EDGE device or devices are responsible for training the deep learning model. For the inference phase, output to a certain query is generated from the EDGE device.

As defined by Zhou et al. [314], All in-EDGE (fifth level) refers to the paradigm where both training and inference of the Deep Neural Network (DNN) takes place in EDGE servers (also known as in-EDGE manner). This level becomes critical when connectivity to the backhaul network becomes limited in the remote location [187]. Another use case when this paradigm becomes essential is when latency is critical for real-time AI applications.

EDGE computing now offers greater computing capability due to significant advancements in the computing power of EDGE servers and the microprocessors used in end devices [172]. Consequently, this improvement in performance means that end devices are now less reliant on Cloud servers for the provision of deep learning inference and training purposes. The all in-EDGE paradigm (EI level 5th) attempts to facilitate the end-user with the necessary computation, storage and latency requirements lessening the dependence on a centralised Cloud server. This research paper primarily focus on All in-EDGE paradigm.

3 DEEP LEARNING AT EDGE

Data generated by end-user devices facilitates the provision of AI-based analytics. More specifically, this data enables us to train deep learning models, which can be used for real-time inference. This section reviews the current state of the art for training deep learning models in the All in-EDGE paradigm. Furthermore, the
section explicitly details the different architecture employed for training and the current enabling technologies and frameworks.

3.1 Architecture

The architecture used for DL training at the EDGE server can broadly be divided into three main categories: centralised, distributed and decentralised, as shown in the Figure 5.

(1) Centralised Architecture:

In a centralised architecture, a single centralised EDGE server (Figure 5(a)) undertakes the DNN training task. Each EDGE server then sends the data produced or accumulated from end devices. Once the centralised EDGE server receives data, it then starts the DNN model training task [111, 204]. In this architecture, the centralised EDGE server is assumed to have enough and relatively higher computing power than other EDGE servers participating in the DNN model training.

(2) Decentralised Architecture:

In a decentralised architecture (Figure 5(b)), each EDGE server is responsible for training their own local DNN. Once a local model is trained, a random peer-to-peer connection with another EDGE server in the network is established for that specific iteration to share their local models. As random sharing of the local model happens after certain iterations, all the EDGE servers achieve a consensus (further updating the model parameters will not change the model’s estimate for a given classification or regression problem). This architecture does not rely on a central authority to manage the DNN training, although the methodology utilised requires an equal contribution from all network servers for the DNN training. Enabling technologies like Aggregation Frequency Control and Gossip training, which supports this architecture, are explained in more detail in section 3.2.

(3) Distributed Architecture:

A distribute architecture is a hybrid approach between the centralised and decentralised architecture. In a distributed architecture, one of the EDGE servers acts as an orchestrator. The rest of the participating EDGE server helps in training the DNN model. Each EDGE server receives the DNN model from the orchestrator to train on their local data. Once local training of the DNN model ends, EDGE servers then send the trained model back to the orchestrator. The orchestrator then combines all the incoming DNN models to produce a global model. There are different techniques that can be applied to combine the incoming models, such as the simple aggregation of model weights. Once the Global DNN model is developed, the orchestrator again sends back the Global DNN model to the rest of the EDGE server for subsequent rounds of training.

3.2 Enabling technologies

This section focuses on the technologies that enable the model training process undertaken by the EDGE Servers. Model parallelism, aggregation frequency control, gossip training, gradient compression, data parallelism, Federated Learning and Split Learning at the EDGE server are the core technologies underpinning centralised, decentralized and distributed architectures. In which Federated Learning and Split Learning have been favoured more in recent years, which is evident from a lot of research interest and citations as presented in Table 3. For comparison, we will be keeping model parallelism, aggregation frequency control, gossip training, gradient compression and data parallelism in one group, and we will call it Group 1. Group 1, Federated Learning and Split Learning technologies are discussed in more detail below and their main differences are highlighted in Table 4. Underneath, we detail the enabling technologies that facilitate deep learning training at the EDGE.

(1) Model Parallelism/ DNN Splitting:

Model Parallelism (also referred to as model splitting or DNN Splitting) is a technique in which the DNN is split across the EDGE servers in order to overcome the constrained computing resources. With the partitioning of the DNN model,
the technique ensures the optimal distribution of the computational workload during the DL training process. Model splitting can be categorised as Horizontal partitioned Model parallelism or Vertical partitioned as shown in Figure 6. In the Vertical Partitioning approach, one or more layers of the DNN are housed in different servers based on the computational requirement of the layer and the available resources of the EDGE server. Whereas, in Horizontal partitioning, neurons from different layers are placed together based on the computational power of the EDGE server.

In [109], the authors proposed STRADS the distributed framework for vertical partitioned parallel machine learning. The ML application scheduler introduced in the STRADS framework helped control the update of the model parameters based on the model’s dependency structure and parameters of the DNN model. The authors also successfully demonstrated 10x faster convergence of the model parallelism based LDA topic modelling implementation over the non-model parallelism. In 2019, research [226] on the training of the Megatron language model also utilised the horizontal partitioned model parallelism for the training of the multi-billion parameter language model. In contrast to the single-GPU-per-model training, authors in this research implemented model parallelism on the same PyTorch transformer implementations with few modifications. To train such a big system, 512 GPUs were consumed to train the transformer-based model. The same model was then able to achieve the SOTA accuracy on the RACE dataset.

(2) Aggregated Frequency Control (AFC): Typically, distributed model training involves: (i) distributing a single copy of a model across multiple edge servers (ii) each of the EDGE servers then train the model locally and (iii) a centralised authority will aggregate the updates from each of the EDGE servers. In AFC, a finite number of discrete clusters of EDGE nodes are formed. The task of each of the discrete clusters is to train an identical DNN model. In each cluster, there is one EDGE server that acts as a parameter server. The task of the parameter server is to provide all other EDGE servers

| Enabling Technology                        | Related Research |
|--------------------------------------------|------------------|
| Model Parallelism/ DNN Splitting           | [67, 87, 109, 158, 226, 284, 287] |
| Aggregation Frequency Control              | [89] |
| Gossip Training                            | [52, 86, 110, 185] |
| Gradient Compression                       | [1, 28, 235, 246, 248] |
| Data Parallelism                           | [27, 121, 125, 131, 193, 241, 298] |
| Federated Learning at EDGE server          | [21, 124, 154, 175, 206, 208, 238, 281, 299, 303, 315] |
| Split Learning at EDGE server              | [3, 63, 75, 189, 250, 251] |
Table 4: Comparison of Enabling Technologies for DL training at EDGE server

| Category                          | Group 1                                      | Federated Learning                                      | Split Learning                                      |
|-----------------------------------|----------------------------------------------|----------------------------------------------------------|-----------------------------------------------------|
| Connected devices                 | Switch, database, access points              | Local database, server, access points, multiplexer       | Bridge, router, switch, database, access points      |
| System Architecture               | Centralized, decentralized and distributed architecture | Distributed architecture                                 | Distributed architecture                             |
| Model                             | Client-server hosted model (as in decentralized) and client-server synergized model (as in centralized and distributed) | Clients server collaboratively trains the model          | Client server shared model architecture and collaborative training |
| Access operational mechanism      | LAN, WAN                                     | LAN, WAN                                                 | LAN, WAN                                            |
| Inter-communication while model training | Full model parameters exchanged (exception Model Parallelism) | Full model parameters exchanged                         | No model parameters are exchanged (only activation vectors from last layers are shared) |
| Computational resource requirement for large DNN | High                                        | High at client and server end                            | Low at client end and comparatively high at server end |
| Data privacy by default           | No                                           | Yes                                                     | Yes                                                 |
| Communication overhead between server and clients | Depends on sample size and model size. | Depends on model size.                                   | Depends on sample size and number of nodes in the cut-layer. |

Figure 6: Model Parallelism.
servers in the cluster with an identical copy of the DNN model. Once each worker node receives their copy, they train that model using their local data and send back the updated DNN model weights to the parameter server for aggregation. The parameter server aggregates the weights from each of the individual nodes in the cluster. Once aggregation is done, the parameter server again sends back the updated DNN model to all the workers in the cluster. In addition, after each aggregation at the parameter server, a “significance function” is computed. This function will determine if the current aggregation has lead to a significant improvement. If the improvement is deemed significant, then the current cluster’s parameter server will inform the parameter servers of each of the other clusters with the new model weights. Hence, each parameter server will have an approximately correct model copy at any given point in time. As shown in Figure 7, Aggregated Frequency Control (AFC) focuses on decoupling the individual nodes’ communication to the centralised authority.

![Figure 7: Aggregated Frequency Control (AFC).](image)

The significance function used in AFC influences the frequency with which updated weights are sent from one parameter server to another. This in turn can reduce the communication overhead in the network. The approximate Synchronous Parallel (ASP) model [89], is one such model that targeted the Geo-Distributed ML training to converge at a faster rate. This research successfully employed an intelligent communication system based on the AFC technique to minimise WAN communication between the two data centres. By utilising the AFC at all-in EDGE paradigm, one can benefit from the low communication between the EDGE servers which are far apart from each other.

(3) Gossip Training:

Gossip Training provides a way to reduce the training time in a decentralised architecture. Gossip training is based on the randomised selection of the EDGE server to share the gradient weights for aggregation [151]. In this technique, each EDGE node will randomly select another node and subsequently send the gradient weight updates to the selected node. Each node will then compute the average received weights. Gossip training is fully asynchronized and works in a fully decentralised manner. In [20], researchers demonstrated that GoSGD (Gossip Stochastic Gradient Descent) takes 43% less time to converge to the same train loss score when compared to EASGD (Elastic Averaging SGD [304]) algorithm used in decentralised architecture training. In other research [212] PeerSGD, modified the GoSGD algorithm [20] to work in the decentralised trustless environment. The algorithm was modified at the stage when the random peer is selected to share the update. The peer who receives the update can decide whether to accept the received weights based on the loss difference (hyper-parameter defined in the research). PeerSGD was evaluated with a varying number of clients ranging from 1 to 100. In the experiment, PeerSGD demonstrated slower convergence when tested with a larger number of clients but it still comparable accuracy after a certain number of epochs. The limitation with PeerSGD is its inability to achieve convergence in a scenario when data classes are segregated across multiple clients. Research [77, 188], targeted Wide Area Network and Heterogeneous EDGE computing platforms with a slightly tweaked version of GoSGD. This work achieved comparable results to the GoSGD algorithm but in different environments. Gossip training provides a way to train a model without a central authority for the all-in EDGE paradigm. Moreover, as EDGE servers can be provided by different institutes that can be trustworthy or not, Gossip training can handle both situations.

(4) Gradient Compression:

Gradient Compression is another approach to reducing communication while training the DL model, which can be applied to either a distributed or decentralised architecture. Gradient Compression (GC) minimises the communication overhead incurred by addressing the issue of redundant gradients. Authors in the research [137], found that 99.9% of the gradient exchange in distributed stochastic gradient descent are redundant. They proposed a technique called Deep Gradient Compression (DGC), which achieved a reduction in the communication necessary for training ResNet-50 from 97 MB to 0.35 MB. In gradient compression, two approaches are used in practice: gradient quantisation and gradient sparsification. In gradient quantisation [246], gradient weights are degraded from having a higher order of precision values to a lower precision order i.e., representing weights using float 12 rather than float64. In [55], the author proposed high-dimensional stochastic gradient quantization for reducing the communication in the Federated Learning setting (explained in 6). In the proposed architecture, the authors decomposed the stochastic gradient into its norm and normalized stochastic gradient. The norm is then compressed via scalar quantizer, and the high-dimensional normalized stochastic gradient is decomposed into two parts. The first part consists of a set of equal-length unitary vectors and a mentioned hinge vector. The unitary hinge vector, when integrated with the first part, yield the normalized stochastic gradient. This unitary nature of both the decomposed part allows them to be compressed using two Grassmannian quantizers. Through the framework of hierarchical gradient quantization, authors reduced the communication overhead.
Data Parallelism:
Data parallelism (also referred to as data splitting) is a technique in which the sizeable primary dataset is split to form mutually exclusive smaller datasets. These datasets are then forwarded to the participating ES (secondary servers). In this architecture, as represented in Figure 8, initially, the primary server distributes the uninitialised similar model copy to each of the secondary servers. The secondary server starts training after the model copy and the associated dataset is received. The primary server holds the responsibility of aggregating the local models residing inside the secondary servers. Once the global model is formed by aggregating the local modal copies, the global model is sent back to the secondary server so that the secondary server can update its local model [125, 182, 241].

Federated Learning:
Federated Learning (FL) is a popular framework for training distributed DNNs [163]. FL provides a practical mechanism to implement DL training across the network hierarchy. Although the native framework treats mobile devices as clients responsible for training the DL model, it can be extended to the EDGE servers [106, 215]. Federated Learning enables EDGE devices and servers (in our case) to collaboratively learn a shared prediction model while keeping all the training data on the device. As shown in Figure 9, during the first stage, all the (client) servers download the global DNN model from the aggregation server (responsible for maintaining the global DNN model). Once the global DNN model is received, client servers train the DNN model on the private data stored in the server, making it a local DNN model. Once training is completed on the client-server, the local model weights are sent to the aggregation server. Once the aggregation server receives all the weights from the participant client servers, it is then aggregated to formulate the new global DNN model [21, 206, 281].

After aggregation, the global DNN model is again circulated to the client servers for further training, making the whole approach cyclic. This framework must ensure that the performance of the aggregated global model should be better than any of the individual client-side models [124] before being disseminated.

Federated Learning Systems (FLS) can be further categorised based on their data partitioning strategy, privacy mechanism and communication architecture [124, 154, 175, 303]. The data partitioning strategy dictates how the data is partitioned across the clients. There are three broad categories of data partitioning (i) Horizontal data partitioned FLS, (ii) Vertical data partitioned FLS and (iii) Hybrid data partitioned FLS. In horizontal data partitioning, all the clients have the same attributes/features in the dataset needed to train the model but use private local data for training. Whereas in vertical data partitioned, all the clients have the different attributes/features in the dataset and then by utilising entity alignment techniques (which helps in finding the overlap...
in different datasets) [208, 238, 315] overlapped samples are collected for training machine learning models. Hybrid data partitioning utilises the best of both worlds. In this category, the entire dataset is divided into horizontal and vertical subsets. So each subset can be seen as an independent dataset with fewer attributes and data points when compared to the entire dataset [303].

For example, a set of hospitals wants to develop the DL model for cancer prediction. Suppose the feature set (medical examination of patients) of a few hospitals matches each other. In that case, they can utilise the Horizontally partitioned scheme, and if the feature set does not match but there is overlap, it can use the vertical data partition scheme [124]. In FLS, even though confidential data is not shared across the client and the server, while exchanging the model parameters, there is the possibility that some sensitive information pertaining to private data could still leaked. The provision of privacy for FLS is typically either through cryptographic methods or differential privacy. When using cryptographic techniques, both the client and server operate on encrypted messages. Two of the most widely used privacy-preserving algorithms are homomorphic encryption [58, 101, 234, 252] and multi-party computation [174, 176, 211, 232]. On the other hand, differential privacy introduces random noise to the data as well as to the model parameters [69, 256, 309, 310]. Although random noise is added to the data, the algorithm provides statistical guarantees on privacy while still ensuring that the data being used facilitates effective model development.

The design of an FLS can be broadly subdivided into two subcategories: distributed and decentralised designs. In a distributed design, a manager is responsible for collecting the local model, aggregating the model and again sending the aggregated global model for retraining. In this design, communication between the clients and aggregation server can happen in synchronous [24, 303] as well as in asynchronous [24, 34, 156, 285] manner. One of the major risks of aggregation servers in a distributed design setting is that the server may not treated each client model equally. That is, the aggregation server may have a bias toward certain clients. A decentralised design can mitigate the potential issues of bias in a distributed design. A decentralized design in Federated Learning can be based on a P2P scheme (ex. gossiping scheme as described in section 3), a blockchain-based system or graph-based system. In decentralized design, none of the participating servers is responsible for being aggregation servers. Therefore, if a gossip scheme is implemented to achieve the decentralized FLS, all the models will randomly share the updates with their neighbours [31, 33, 149]. In contrast, if a blockchain system is implemented, it leverages Smart Contracts (SC) to coordinate the round delineation, model aggregation, and update tasks in FLS [130, 152, 184, 203, 254]. Lastly, if graph-based FLS is implemented, each client will utilize the graph neural network model with its neighbours to formulate the global models [15, 84, 143, 283].
across the clients and the servers. Figure 11 shows three configurations—simple vanilla Split Learning, Split Learning without label sharing and Split Learning for vertically partitioned data. A main neural network is partitioned into two sub-networks in simple vanilla Split Learning. The initial sub-network, along with the input data for the neural network, remains with the client, whereas the remaining sub-network, along with the labels, reside with server [251]. Split Learning without label sharing is identical to the vanilla Split Learning, except that the labels live with the client instead of the server. To compute the loss, the activations outputted from the server-side network are sent back to the client, who holds the last layer of neural network [3, 63, 75]. The loss is calculated, and gradients are computed from the last layer held by the client and then sent back to the server, and backpropagation takes place in the usual way. Clients train their partial sub-network for vertically partitioned data and then propagate the activation to the server-side sub-network. The server-side sub-network then concatenates the activation’s and feed them to the remaining sub-network. This configuration, labels are also shared with the server [278]. In a Federated Learning system, clients can interact with the server in parallel, which helps training to be faster than a Split Learning-based system. In contrast to Federated Learning, Split Learning provides a better means to reduce the computational requirements on the client-side. This reduction in computation comes from the fact that instead of training the whole neural network at the client-side (as done in a Federated Learning system), now the client has to compute only a sub-network of the whole network (as done in a Split Learning system). Recently, to leverage the advantages of both Split Learning and Federated Learning a hybrid techniques called SplitFed Learning was proposed [250]. In splitfed learning, a neural network is broken down into the sub-networks shared amongst the clients and servers. In addition there is a separate federated aggregation server for the client and for the servers. All the clients perform the forward pass in parallel and independent of each other (as seen in Split Learning). The resulting activations are sent to the server-side sub-network, which performs a forward pass for the remaining sub-network portion. The server then calculates the loss and back propagates the gradients back to the very first layer on the client-side (as described earlier with Split Learning). Once this process finishes, the servers send their model weights to a federated aggregation server, which aggregates the independent server-side sub-network to form a global server-side model. Similarly the clients send their sub-network weights to another aggregation, at the end of aggregation, a global model can be developed by combining the aggregated client side weights with the aggregated server side weights as show in Figure 12 (a) [189, 250]. Splitfed learning can have several variants. For example, the first one is where each client has its own corresponding server-side network in the main server, i.e., number of client-side models are equal to the number of server-side models as explained in the earlier paragraph. In the second variant there are multiple client but only a single server. Therefore, each client-side model send their activations to the a single common server-side sub-network, thereby reducing the required aggregation step and need to keep the multiple copies of the server-side networks as compared to the first variant as shown in Figure 12 (b). Moreover, as the server keeps only one copy of the server-side sub-network, it makes the server-side do forward and backward pass sequentially with each of the client’s data (activations of the cut layer) [66, 103].

3.3 Model Adaption at EDGE server

Model Adaption techniques provide a means by which DNN deployment at the EDGE server can deliver high-quality EI services despite a lack of computing resources, storage, and bandwidth. Model adaption techniques can be broadly categorized into model compression and conditional computation techniques, as summarised in Table 5.
Figure 12: Variants of splitfed learning (a) Splitfed learning with same number of client and server side sub-networks and (b) Splitfed learning with only one copy of server-side sub-network.

Table 5: Enabling Technologies for model adaptation at EDGE server

| Model Adaptation Category | Model Adaptation Technique |
|---------------------------|----------------------------|
| Model Compression         | Pruning [19, 71, 78, 79, 132, 134, 140, 171, 200, 265, 288] |
|                           | Quantization [19, 68, 90, 134, 166, 179, 291, 307, 311, 314] |
|                           | Knowledge Distillation [29, 41, 42, 80, 88, 95, 170, 186, 195, 220, 228, 231, 239, 257, 266, 267, 274, 295, 296, 300, 312] |
|                           | Low rank factorization [76, 98, 119, 168, 190, 196, 210, 292] |
| Conditional Computation   | Early Exit [12, 118, 126, 161, 194, 244, 245, 249, 271, 282, 313] |
|                           | Model Selection [159, 191, 271, 314] |
|                           | Result Cache [13, 39, 53, 92, 93, 96, 108, 112, 114, 123, 209, 268, 293, 319] |

3.3.1 Model Compression: Model compression techniques facilitate the deployment of resource-hungry AI models into resource-constrained EDGE servers by reducing the complexity of the DNN. Model compression exploits the sparse nature of gradients’ and computation involved while training the DNN model. In turn, this has been shown to reduce network latency, memory and energy. This section reviews pruning, quantisation, knowledge distillation and low rank factorization.

(1) Pruning:

Pruning of parameters is the most widely adopted approach to model compression. In this approach, neural network parameters are evaluated against their contribution to predicting the label. Those neurons that make a low contribution in inference are then pruned from the trained DNN. Pruning of parameters reduces the size of a DNN but can also negatively impact network performance. In [79] the research authors were able to reduce the size of the AlexNet and VGG-16 by a factor of 9x and 13x respectively, without incurring any loss in the accuracy over the ImageNet dataset. In other research [78], the authors utilised pruning to create a compressed speech recognition model on field-programmable gate-array (FPGA). SS-Auto [132] is a single-shot structured pruning framework. In contrast to earlier versions of pruning where whole DNN network parameters were selected for pruning, in structured pruning, an independent pruning on columns and rows of filters and channels matrix (for CNN based DNN model) is performed. The compressed DNN model produced by the SS-Auto framework did not suffer any degradation in performance, achieving the original perform levels when tested on CIFAR-10 and CIFAR-100 datasets. However, the compressed VGG-16 model reduced the number of convolutional layers parameters by a factor of 41.4% for CIFAR-10 and 17.5% for CIFAR-100 dataset. In [71], the authors proposed a new framework based on weight pruning and compiler optimisation for faster inference while preserving the privacy of the training dataset. This approach initially trains the DNN model as usual on the user’s own data. The model then undergoes privacy-preserving-oriented DNN pruning. Synthetically generated data (with no relevance to the training data) is passed through a layer of the user-trained model. The decision to prune a parameter or not from the current layer is based on how similar (by computing the Frobenius norm) the original output of the layer (without pruning) when compared with the output of the
layer after the parameter has been pruned. If the outputs are close enough, then that parameter is pruned. This pruning technique is named as alternating direction method of multipliers (ADMM). Experimental results of the framework outperformed the state-of-the-art end-to-end frameworks, i.e., TensorFlow-Lite, TVM and MNN, with speedup in inference up to 4.2×, 2.5×, and 2.0×, respectively.

(2) Quantisation:
Data quantisation degrades the precision of the parameters and gradients in the DNN. More specifically, in quantisation, data is represented in a more compact format (low precision form). For example, instead of adopting a 32-bit floating-point format, a quantisation approach might utilise a more compact format such as 16-bit to represent layer inputs, weights, or both [314]. Quantisation reduces the memory footprint of a DNN and its energy requirements. In contrast, pruning of the neurons in a DNN will reduce the networks memory footprint but does not necessarily reduce energy requirements. For example, if later stage neurons are pruned in a convolutional network, this will not have a high impact on energy because the initial convolutional layer dominates energy requirement [314]. In [291], the authors utilised a dynamic programming-based algorithm in collaboration with parameter quantisation. With the proposed dynamic programming assisted quantisation approach, the authors demonstrated a 16× compression in a ResNet-18 model with less than a 3% accuracy drop. The authors in [90], proposed a quantisation scheme for DNN inference that targets weights along with the inputs to the model and the partial sums occurring inside the hardware accelerator. Experiments showed that the proposed schema reduced the inference latency and energy consumption by up to 3.89% and 4.84× respectively while experiencing a 1.18% loss in the DNN inference accuracy.

(3) Knowledge Distillation:
The knowledge distillation model compression technique is composed of three key components: Knowledge, the distillation algorithm, and teacher-student architecture [72]. Knowledge is the representation learnt by the teacher model. It is usually represented by a large neural network trained on a large amount of data. The knowledge distillation algorithm is used to transfer the Knowledge from the teacher model to the student model such as Adversarial KD [170, 257], Multi-Teacher KD [80, 267, 300], Cross-modal KD [41, 228, 239], Attention-based KD [29, 95, 195, 296], Lifelong KD [42, 295, 312] and Quantized KD [22, 102]. Finally, the teacher-student architecture is used to train the student model. A general teacher-student framework for the Knowledge distillation is shown in Figure. 13. In this architecture, the teacher DNN model is trained on the given dataset in the initial phase. Once the teacher DNN model is trained, it then helps the shallower student DNN model. The student DNN model also uses the same dataset which was used to train the teacher DNN model, but labels for the data points are generated by the teacher DNN model [165]. Thereby, the knowledge distillation technique helps a smaller DNN model imitate the larger DNN model’s behaviour.

Figure 13: Teacher-student architecture for Knowledge Distillation.

KD provides a viable mechanism of model compression [72, 91]. However, a mismatch in the accuracy during model evaluation indicates students’ incapability to mimic the teacher perfectly [40, 72, 308], which requires more research in future.

(4) Low Rank Factorization:
Low rank factorization is a technique which helps in condensing the dense parameter weights of a DNN [98, 190], limiting the number of computations done in convolutional layers [76, 119, 168, 196] or both [210, 292]. This technique is based on the idea of creating another low-rank matrix that can approximate the dense metrics of the parameter of a DNN, convolutional kernels, or both. Low-rank factorisation can save memory on EDGE servers, while also decreasing computational latency because of the resulting compacted size of the models. In [37], the authors used the low rank factorisation by applying a singular value decomposition (SVD) method. They demonstrated a substantive reduction in the number of parameters in convolutional kernels, which helped reduce floating-point operations (FLOPs) by 65.62% in VGG-16 while also increasing accuracy by 0.25% when applied to the CIFAR-10 dataset. Unlike pruning, which necessitates retraining the DNN model, after the application of low-rank factorisation there is no need to retrain the DNN model. Further research [240] proposed a sparse low-rank approach to obtain the low-rank approximation. The sparse low-rank approach is based on the idea that all the neurons in a layer have different contributions to the performance of the DNN model. So based on the neuron ranking (based on the contribution made for inference), entries in the decomposition matrix were made. This approach, when applied over the CIFAR-10 dataset with VGG-16 architecture, achieved 3.6× times smaller compression ratio to the SVD. Other most commonly used methods for Low rank factorization are Tucker Decomposition (TD) [62, 155, 222] and canonical polyadic decomposition (CPD) [25, 197].

3.3.2 Conditional Computation: Conditional computational approaches alleviate the tension between the resource-hungry DNN model and the resource-constrained EDGE servers. In conditional computation, the computational load of the DNN deployed over a single ES is distributed across the network hierarchy. The selection of the appropriate conditional computation technique is based on the EI application’s latency, memory, energy, and network requirements. Therefore, depending upon the configuration of the
ES and its application requirements, one or any combination of the following techniques (Early Exit, Model Selection and or Result Cache) is employed to empower high-quality EI services.

(1) Early Exit:
The main idea behind the early exit approach is to find the best tradeoff between the deep DNN structure and the latency requirements for inference. In this approach, a deep neural network trained on a specific task is partitioned across the EDGE servers. The partitioning of the DNN model is based on a layer-wise split, such that a single or multiple layers can reside in the ES based on the computation power provided by that ES. Each ES that hosts one or more layers of the DNN also attach a shallower model (or side branch classifier) to the output of the final layer on the current ES. The model is then trained as shown in Figure 14. The purpose of the side branch classifier is to provide an early prediction or early exit. During inference the data is propagated through the network (and each ES host). Each host will calculate both the output of the hosted layers and the output of the local early exit network. If the output of the early exit layer exceeds a defined confidence threshold then the propagation stops (this is the early exit) and the ‘early’ result is returned. In the case that the prediction from the early exit network is less than the confidence threshold the output of the larger DNN layers is then propagated to the next ES in the chain, which holds the next layer of the larger DNN and another early exit network. The process of propagating the layer’s output to the subsequent layer is carried out until one ES infers the class with a higher confidence score. This process can provide ‘n-1’ exit points for a DNN with an ‘n’ deep-layer structure. Thus, if layer 1 of larger DNN along with the side branch can infer the class with required confidence that output will be given as a response to the end-user eliminating any further propagation of activation values along the ES.

Figure 14: Early exit adaption of Deep Neural Network.

Researchers in [249], provided the programming framework ‘Branchynet’, which helps incorporate the early exit approach into a standard DNN. The framework modifies the proposed DNN by adding exit branches at certain layers. With the multiple early exit points, it can also be considered as an enabler for localized inference using shallow DNN models [271]. In [126], the authors proposed DeepQTMT to lower the encoding time spent on video compression. In the DeepQTMT, the authors utilised a multi-stage early exit mechanism to accommodate the high encoding time. Experimental results showed the encoding time was reduced by a factor of 44.65% - 66.88% with a negligible delta in bit-rate of 1.32% - 3.18%. With the early exit strategy, one can benefit from low latency as a faster response for the user query. The drawback of the early exit technique is that it increases the memory footprint of the DNN, thus utilising more storage at each individual EDGE server.

(2) Model Selection:
The model selection approach selects a specific DNN model for inference from a set of available DNN models based on the latency, precision and energy requirements of the end-user [271]. In a model selection strategy, multiple DNN models with varying DNN structures are trained. The different trained models each have a specific inference latency, energy requirements, and accuracy. Once trained, each of the models is deployed to various servers. The model selection approach will then select the DNN model based on the end-user requirements [314]. The model selection approach is similar to the early exit approach with only one difference, that, in model selection, independent DNN models are trained. In contrast, in the early exit, only one DNN is trained over which multiple exit points are created. Authors in [191], proposed a new concept of BL-DNN (big/Little DNN) based on the model selection approach. The authors proposed the score margin function, which helps in taking the decision whether or not inference made by small DNN is valid or not. The score function is computed by subtracting the probability of the first prediction with the second prediction for the given input. Thus, a score function can be seen ranging from 0 to 1. Higher the value of the score function, the higher the estimation that inference is accurate. Lower the value of score function lower is the estimation of inference being accurate. If the score function estimation is low, then a big DNN is invoked to make the inference on the same input data. The same research was able to show a 94.1% reduction in the energy consumption on the MNIST dataset with accuracy dropping by 0.12%. Recently in [159], an adaptive model selection technique is used to optimise deep learning inference. The proposed framework builds a standard machine learning model, which learns to predict the best DNN model to use for inference based on the input feature data. To facilitate the training of the selection model (which is standard KNN model in this scenario), different pre-trained models like Inception [148], Resnet [217], MobileNet [104] were evaluated on the same image dataset. For each image, the DNN model that achieved the highest accuracy is set as the output. The training data for the KNN model is comprised of the features extracted from the image as input and the optimal DNN model as output. Once the model
selector (the KNN) is trained, it is then used to determine the DNN model, giving the best accuracy on the selected image. In the end, the selected DNN model makes inference on the image as shown in Figure 15.

![Diagram](image)

**Figure 15: Model Selection of Deep Neural Network.**

Experimental results validated the reduction in the inference time by a factor of $1.8 \times$ for the classification task and $1.34 \times$ time reduction in a machine translation task. Model selection facilitates a decrease in inference time. However, with an increase in the number of pre-trained DNN models, the memory footprint across the EDGE servers increases significantly.

(3) Result Cache:

Result cache techniques help in decreasing the time required to obtain the prediction from the EDGE server. In this approach, frequent input queries (such as frames in the case of video classification or images in the case of image classification) and associated DNN predicted output are saved in an archive on the EDGE server. So, before any query is inferred from the DNN model, intermittent lookup happens. In intermittent lookup, if a query is similar to a saved query, the result is inferred from the archive (Cache). Otherwise, the query goes to the DNN model for inference. This technique becomes more powerful in environments where the queries can be expected to exhibit similarity. Drolia, Utsav, et al. proposed a cache-based system that leveraged the EDGE server for image classification [53]. When evaluated on image classification applications, the approach yielded up to $3 \times$ speedup on inference for image recognition task without any drop in the model’s performance (accuracy). Another system for video analysis utilised the cached convolution outputs of the CNN layers to reduce the computation for making an inference [92]. The idea is again based on the similarity of consecutive frames in videos. Initially, in this approach, activations from each layer of DNN for a query frame are saved in the cache. For the next subsequent frame(query), the query is pushed through the first layer and the resulting activations are compared with the previous activation values of the same layer saved in the cache. Only those activations that differ significantly from the cached version are calculated and propagated further through the network. If the activation is deemed similar, they are carried over with their cache results to the next layer. In the experiment, the authors showed a significant speedup of $3 \times$ to $4 \times$ compared to the vanilla CNN model with no change in accuracy. In other research [13], the authors proposed the framework on a similar line of result caching. In this research, queries were initially passed through the DNN and activation’s of each layer were cached (archived) in the EDGE server along with the prediction from the DNN model. During the inference, after passing the image through the layers of DNN, activation’s are then checked with the saved activations of a specific layer. If activation of a particular layer for query matches with the activation in the cache, further propagation of activation is stopped, and the cached result is provided back for the query. The research with VGG-16 architecture on CIFAR yielded a 1.96x latency gain using a CPU and a 1.54x increase when using a GPU with no loss in accuracy. Result caching provides a significant boost in the scenario where the query (frames processing for the boundary identification) for inference does not change significantly. While result caching improves the overall latency of the neural network, it also incurs a larger memory footprint.

4 KEY PERFORMANCE METRICS FOR AI AT THE EDGE

The application of AI at the EDGE has gathered significant momentum over the last few years. Consequently, significant changes can be seen in the selection of evaluation metrics for EI based services. Selection of the performance metrics for EI services is dependent on latency, bandwidth (data transferred across the backhaul network), privacy and storage requirements. Also, some additional criteria need to be considered and monitored when training AI models at the EDGE.

This section will discuss the different evaluation metrics that should be evaluated when developing All-in EDGE based AI models.

4.1 Use-case specific metrics

Use-case specific metrics are used to determine the quality of the trained DL model and are dependent on the problem statement. For example, if the use-case is a classification problem, then accuracy, balanced accuracy, roc_auc, etc. can be evaluated [113, 229, 297]. Simultaneously, if it is a regression-based use-case, one needs to assess max variance, R-square, root mean squared error, etc. [59, 115, 177, 224, 263]. While these metrics are widely used, they are essential for the performance comparison of different models architecture and strategies deployed on the same dataset over the EDGE server.

4.2 Training Loss

The process of training a deep learning model requires the optimisation (typically minimisation) of a specific loss function. The training loss is a metric that captures how well a DNN model fits the training data by quantifying the loss between the predicted output and ground truth labels. Different metrics are selected based on the type of problem, i.e., classification or regression. Some of the widely used loss functions to capture the learning of DNN at EDGE while training are Mean Absolute Error [60, 64, 273], Mean Square Error
When a DNN model is deployed for inference on an EDGE server, inferring from a model at the EDGE both the computational and communication latency are more favoured. This metric becomes critical as one of the most important reasons to move from Cloud to All-in EDGE was to reduce the latency incurred during DNN training and inference.

4.5 Communication Cost

When a DNN model is deployed for inference on an EDGE server, many requests by the end-user(s) are raised to consume the EI service. The volume of data transmitted from the end-user(s) has the potential to create congestion at the network EDGE server. The communication cost metric evaluates the amount of data (message size of each query) flowing to the ES from the end-user [135, 223]. It also takes into consideration the inference data, which is reverted to the end-user. Active monitoring of the communication cost is important to ensure concise data flow and prevent any potential congestion points [99, 253, 275].

4.6 Privacy-preserving Metrics

Privacy-preserving metrics provide a means to quantify the freedom of privacy that is enjoyed by users when privacy protection is offered by an EI application by privacy-preserving technologies [264]. In order to assess the merits of the enabling technologies to preserve privacy, it can be evaluated on its merits to minimize:

- Direct leakage
- Indirect leakage

Direct leakage can occur during the training process when the privacy of the training data can be compromised or at the inference time when the client’s data gets hacked before reaching the EI application. Protecting the client’s data at inference time is managed by well-established encryption algorithms like DES, 3DES, AES, RSA and blowfish, which doesn’t require evaluation [259]. However, we need to measure the privacy of training data when used in building an AI model. Based on the enabling technology utilised to preserve privacy, there will be different metrics that can help to evaluate it. Enabling technologies, where activations are transferred from one server to another, can leak private data. Enabling technology such as model parallelism, gradient compression, and Split Learning-based systems to preserve privacy minimises the similarity between the raw data and the intermediary activation vector sent from one server to another. The distance correlation metric such as pairwise correlation and mutual information score [242, 261, 262] can be utilised to find the leakage between the raw data and intermediary activation vector. This metric ranges from 0 to 1, where 0 implies the raw data are independent of the intermediary activation vector.

Similarly, indirect leakage happens when servers share model parameters during the training process. This exposes the client’s dataset to inference attack by adversaries [270]. Enabling technologies such as aggregation frequency control, gossip training, data parallelism, and Federated Learning frequently shares the model parameters with other participating servers. It makes them prone to leak client data from the model parameters. Mutual information as a metric provides a way to quantify the risk of confidential information leakage from the gradients [32]. Mutual information quantifies the amount of common information obtained about the client’s data by observing the model parameters [146].

4.7 Energy consumption

There is a wide range of available DNN models, their individual energy requirements for computation can vary significantly. For some resource-constrained environments, it becomes infeasible to host models with a larger energy footprint [48, 49]. The energy consumption of different models should be evaluated to find the best DNN architecture deployment strategy. The energy requirement of a DNN model should be considered for both the training and inference phase [164, 183, 317]. Power consumption (watts and kilowatts units) as measurement can be utilised to determine the energy consumption [133].

4.8 Memory footprint/ Model Size

As an EDGE server usually has limited infrastructure resources, it becomes challenging to host a DNN model because of the computational requirements (the bigger the network, the more parameters it will have and each extra parameter increases the memory requirement (in RAM)). Model size or memory footprint is computed having ‘MB’ as their unit of measurement [35, 61, 139, 167, 260]. For the image classification problem, if MobileNet V2 with 3.54 million parameters is selected it will have 14 MB as model size whereas if InceptionV4 with 42.74 million parameters is selected for the same problem it will have a 163 MB model size requirement [133].
5 OPEN CHALLENGES AND FUTURE DIRECTION

Thus far, we have discussed deep learning architectures, techniques, adaption techniques and the key performance indicators required to facilitate AI to the All-in EDGE paradigm. In this section we will highlight existing open challenges and future research directions in the area of AI at the All-in EDGE paradigm.

5.1 Latency

The deployment of AI at the EDGE server enables low latency inference due to the closer proximity to the end-user. AI applications like image segmentation demand very low latency to be practical in real-world applications. Researchers from academia and industry are actively pursuing methods of decreasing latency by utilising EDGE based AI models [94, 107, 129, 157, 202]. The research focused on deploying models at the EDGE has exclusively focused on using CNN-based DNN models. Unfortunately, to date, a host of other types of DNN models have not been considered. For example, open areas of research include the EDGE deployment of models with looped layers, such as RNNs and larger scale models such as transformers. Deployment of such models will benefit in building better applications based on information retrieval, language model, object detection, image segmentation etc.

5.2 Memory efficiency

DNNs are resource-hungry during the model training and inference stages. One of the significant challenges at the EDGE server is the availability of limited computing resources. EDGE servers share computing resources across multiple applications and resource-hungry DNN applications have the potential to impact the normal operation and available resources for other applications running at the EDGE server. To address this issue, efforts have been made to update the hardware chipset [198] to facilitate higher processing rates. Other research has examined methods of improving communication efficiency (to pass data only when it’s really necessary from one EDGE server to another) [218]. Research needs to find approaches that can reduce the memory footprint of DNN at EDGE server by utilising model adaption and improved communication efficiency to mitigate the memory-efficiency requirement in the end-to-end EI services.

5.3 Privacy-preservation in EDGE-AI

Providing adequate privacy preservation for EI applications is an area with open research challenges. To preserve the privacy of client’s data, different enabling technology are utilised with or without cryptographic techniques, perturbation techniques, and anonymisation techniques [227, 255]. On the one hand, these technologies and techniques provide a means to safeguard the client’s data better but simultaneously struggles to maintain the effectiveness (accuracy in case of classification problem) of the AI model [7, 65, 105]. On the other hand, it’s not effectiveness that gets jeopardised, but efficiency (which includes training time [251, 305] and inference time [97, 243]) of the AI model also gets degraded. Hence, there lies an opportunity to build a platform that can preserve privacy but at a considerable cost of sacrificing the effectiveness and efficiency of the AI model.

5.4 Designing EI Application Framework for All-in EDGE paradigm

All-in EDGE paradigm requires new ways of designing applications. In section 3.1, we presented different architectures capable of pushing AI to the EDGE server with varying application requirements. With the enabling technologies (model parallelism, aggregation frequency control, gossip training, gradient compression, data parallelism, Federated Learning and Split Learning as explained in section 3.2) and model adaption techniques 3.3, EI application design becomes progressively more complex. The introduction of a microservices-based architecture is another exciting area of research in the provisioning of EI applications [56]. Although different research provided the framework, they all remain confined to the problem they tried to resolve. For example, in [280] provided a framework for self-learning EI, in which authors proposed GAN based synthesis of the traffic images. The proposed framework remains applicable for only video-based scenarios.

Similarly, the work in [10] provides a framework that was restricted to work for the web traffic anomaly detection. Likewise, other research [120, 136, 141] have their own niche and the proposed framework is restricted to solve the specific problem type. To the authors knowledge, Open EI [306], is the only framework that provided a generic approach to facilitate the development of applications for a wide range of problem domains (computer vision, natural language processing etc.). Still, this framework lacks the components of hardware (choices in the selection of hardware accelerators that can help in faster DNN computation i.e., [81, 173, 221, 301]) and the deployment of the EI services (how to distribute load and develop a global model across the EDGE servers 3.2). Therefore, there is a need to find a robust framework that can help in the easy deployment of complex EI architecture while finding the best trade-off between the application requirements and the EDGE server resources.

6 CONCLUSION

Exploding data due to the proliferation of EDGE devices and advancements in resource-hungry Deep Learning (e.g., Deep Neural Network) models lead to new challenges that need to be considered to enable Deep Learning in the All-in EDGE paradigm. In this regard, this survey paper focused on the current state-of-the-art facilitating Deep Learning in the All-in EDGE paradigm. We initially performed a thorough review of the various levels of EDGE Intelligence. We subsequently focused on the All-in EDGE paradigm, the motivation behind the adoption of EDGE Intelligence. We presented an overview of the architectures, enabling technologies and model adaption techniques that enable EDGE intelligence through Deep Learning. Then, we presented the key performance metrics that should be tracked to analyze the All-in EDGE services and Deep Learning techniques at the edge. Finally, we highlighted open challenges and future research directions.

ACKNOWLEDGMENT

This research was conducted with the financial support of ADVANCE CRT PhD Cohort under Grant Agreement No. 18/CRT/6222 and at the ADAPT SFI Research Centre at Cork Institute Of Technology. The ADAPT SFI Centre for Digital Media Technology is
funded by Science Foundation Ireland through the SFI Research Centres Programme and is co-funded under the European Regional Development Fund (ERDF) through Grant 13/RC/2106.

REFERENCES

[1] Lusine Abrahamyan, Yiming Chen, Giannis Bekoulis, and Nikos Delfounnis. 2021. Learned Gradient Compression for Distributed Deep Learning. arXiv preprint arXiv:2103.08870 (2021).

[2] Haftay Gebreselasie Abera, Mohammad Hayajneh, and Mohamed Adel Serhani. 2022. Federated Learning in Edge Computing: A Systematic Survey. Sensors 22, 2 (2022), 450.

[3] Sharif Abudabbba, Khyuoyeen Kimi, Minki Kim, Chandra Thapa, Seyit A Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. Can we use split learning on 1d mm models for privacy preserving training? In Proceedings of the 11th ACM Asia Conference on Computer and Communications Security. 305–318.

[4] Karan Aggarwal, Maad M Mijwil, Abdel-Hameed Al-Mistarehi, Safwan Alomari, Isiaka A Alimi, Romil K Patel, Aziza Zaouga, Nelson J Muga, Armando N Pinto, and Christophe Blanc. 2021. Deep model compression and architecture optimization for embedded systems: A survey. Distributed Cloud, Fog And Edge Computing and Their Opportunities and Challenges. Research.Ber. Review of Science and Technology. 1, 1 (2021), 1–15.

[5] Munirah Alkhelaiwi, Wadii Boulila, Jawad Ahmad, Anis Koubaa, and Maha Drias. 2021. An efficient approach based on privacy-preserving deep learning for satellite image classification. Remote Sensing 13, 11 (2021), 2221.

[6] Abdallah Abulseumi, Yassine Himeur, Faycal Bensaihi, and Abbes Amira. 2022. An innovative edge-based Internet of Energy solution for promoting energy saving in buildings. Sustainable Cities and Society 78 (2022), 105571.

[7] Syed Umar Amin and M Shami Hossain. 2020. Edge Intelligence and Internet of Things in Healthcare: A Survey. IEEE Access (2020).

[8] Yufei An, Jianqiang Li, F Richard Yu, Jianyong Chen, and Victor CM Leung. 2020. Efficient resource allocation in MEC. arXiv preprint arXiv:2007.06081 (2020).

[9] Defang Chen, Jian-Ping Mei, Yuan Zhang, Can Wang, Zhe Wang, Yan Feng, and Cnnn Chen. 2021. Cross-layer distillation with semantic calibration. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. 7028–7036.

[10] Yijun Chen and Xukan Ran. 2021. Deep learning with edge computing: A review. Proc. IEEE 109, 8 (2021), 1655–1674.

[11] Jin-Hua Chen, Min-Rong Chen, Guo-Qiang Zeng, and Ji-Si Weng. 2021. BDFL: a byzantine-fault-tolerance decentralized federated learning method for autonomous vehicle. IEEE Transactions on Vehicular Technology 70, 9 (2021), 8639–8652.

[12] Anandkumar. 2018. signSGD: Compressed optimisation for non-convex problems. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32. 305–313.

[13] Masaaki Basu, Rodolfo Lizcano, Christian Hwang, and Marcello Pedocchi. 2021. GitDNN: A platform for measuring and evaluating distributed deep learning systems. arXiv preprint arXiv:2105.14408 (2021).

[14] Yaming Chen, Xiao Jin, Xuejiao Sun, and Xiaoqian Yin. 2020. VaF: a method of vertical asynchronous federated learning. arXiv preprint arXiv:2007.06081 (2020).

[15] Yung-Sung Chuang, Shang-Yu Su, and Yun-Nung Chen. 2020. Lifelong language knowledge distillation. In Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. 6794–6802.

[16] Takanori Chikada, Hiroshi Abe, and Kaoru Nakamura. 2021. An efficient flow for federated learning. arXiv preprint arXiv:2105.06250 (2021).

[17] Toshiyuki Cho, Hirosuke Ishihara, and Junya Nakatani. 2020. An efficient cross-model distillation. arXiv preprint arXiv:2005.08211 (2020).

[18] Robert Dale. 2021. GPT-3: What’s it good for? Natural Language Engineering 27, 1 (2021), 113–118.

[19] Mario AR Dantas, Paulo Eduardo Bogoni, and Paulo Jose De Freitas Filho. 2020. An application study case tradeoff between throughputs and latency on fog-cloud cooperation. International Journal of Networking and Virtual Organizations 25, 3 (2020).
Radosvet Desislavov Georgiev. 2021. Analysis of Deep Learning Inference Dimensions.ai. 2022. Publications Overview. https://tinyurl.com/mt9s9fyr

Umar Farooq, Muhammad Wasif Shabir, Muhammad Awais Javed, and Muhammad Imran. 2021. Emanuel Figetakis and Ahmed Refaey. 2021. UAV Path Planning Using an On-Board Low Rank Tucker Decomposition for 2D+3D Facial Expression Recognition. Electronics 10, 12 (2021), 1–24.

Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with dynamic model-parallel solution to DNN training. In Proceedings of the 10th Annual Conference on Scientific Cloud Computing (SCC'2020).

Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Heye Zhang, Shizhou Dong, Shanhui Sun, Xin Wang, Guang Yang, Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, et al. 2021. Efficiency Privacy Preserving Edge Intelligent Computing Framework for the Deployment of Edge Inference: The Conclusions to Start Designing. IEEE, 1–6.

Chen Chen. 2020. Accelerating Gossip-based Deep Learning in Heterogeneous Edge Computers. In Proceedings of the 10th International Conference on Cloud Networking (CloudNet).

Yao Jiong Hu, Soyi Jung, Jae-Hyun Kim, Marco Leverato, and Joanheung Kim. 2021. Spatio-Temporal Split Learning for Autonomous Aerial Surveillance using Urban Air Mobility (UAM) Networks. arXiv preprint arXiv:2111.11856 (2021).

Han, Junlong Kang, Huizi Mao, Yiming Hu, Lin Xi, Yubin Li, Dongliang Xie, Hong Luo, Song Yu, Wu Yang, et al. 2017. Ese: Efficient speech recognition engine with sparse latents on fpga. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 75–84.

Song Han, Jeff Pool, John Tran, and William Dally. 2021. Learning Versatile Convolution Filters for Efficient Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning. Future Internet 13, 4 (2021), 94.

Abu Ahsan, KN Bhanu Prakash, Viraj Kulkarni, and Aniruddha Pant. 2021. Comparison of Privacy-preserving distributed deep learning methods in healthcare. arXiv preprint arXiv:2112.15291 (2020).

Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Heye Zhang, Shizhou Dong, Shanhui Sun, Xin Wang, Guang Yang, Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, et al. 2021. Efficiency Privacy Preserving Edge Intelligent Computing Framework for the Deployment of Edge Inference: The Conclusions to Start Designing. IEEE, 1–6.

Hanieh Hashemi, Yongqiang Wang, and Murali Annamraju. 2021. DarkKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In MICRO-54. 54th Annual IEEE/ACM International Symposium on Microarchitecture, 212–224.

Najmul Hassan, Saira Gillani, Ejaz Ahmed, Ibrar Yaqoob, and Muhammad Imran. 2018. The role of edge computing in internet of things. IEEE communications magazine 56, 11 (2018), 110–115.

Bin Ren, Caiwen Ding, Xue Lin, Xiaolin Xu, et al. 2020. A Privacy-Preserving-Trustable Hardware. In Proceedings of the 2020 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI). IEEE, 1–5.

Fang Han, Junlong Kang, Huizi Mao, Yiming Hu, Lin Xi, Yubin Li, Dongliang Xie, Hong Luo, Song Yu, Wu Yang, et al. 2017. Ese: Efficient speech recognition engine with sparse latents on fpga. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 75–84.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning Versatile Convolution Filters for Efficient Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning. Future Internet 13, 4 (2021), 94.

Umar Farooq, Muhammad Wasif Shabir, Muhammad Awais Javed, and Muhammad Imran. 2021. Intelligent energy prediction techniques for fog computing networks. Applied Soft Computing 111 (2022), 106782.

Emmanuel Figetakis and Ahmed Refaey. 2021. UW Path Planning Using on-Board Ultrasound Transducer Arrays and Edge Support. In 2021 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 1–6.

Georgios Flamis, Stavros Kalapothas, and Paris Kitsos. 2020. Best Practices for the Deployment of Edge Inference: The Conclusions to Start Designing. Electronics 10, 16 (2021), 1912.

Yunfang Fu, Yujian Deng, Yuehu Zhuang, Zhengyan Yang, and Ruli Qi. 2022. Low Rank Tacker Decomposition for 2D+3D Facial Expression Recognition. Procedia Computer Science 198 (2022), 499–504.

Yansong Gao, Minshi Kim, Sharif Abudubbia, Yeonjae Kim, Changha Thapa, Kyunong Kim, Sehye A Castephe, Hongyuchang Kim, and Surya Nepal. 2020. End-to-end evaluation of federated learning and split learning for internet of things. arXiv preprint arXiv:2005.13376 (2020).

Zhihua Wu. 2021. From Server-Based to Client-Based Machine Learning: A Comprehensive Survey. ACM Computing Surveys (CSUR) 54, 1 (2021), 1–36.

Zhihua Xu, Mingrui Liu, Zhongming Yuan, Li Shen, Wei Liu, and Tianbao Yang. 2020. Communication-efficient distributed stochastic aac maximization with deep networks. In International Conference on Machine Learning, PMLR, 3864–3874.

Yoo Jeong Ha, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. 2020. Low Rank Tucker Decomposition for 2D+3D Facial Expression Recognition. arXiv preprint arXiv:2003.13376.
Lingjing Kong, Tao Lin, Anastasia Koloskova, Martin Jaggi, and Sebastian U Stich. 2021. Consensus Control for Decentralized Deep Learning. arXiv preprint arXiv:2102.04828 (2021).

Xiangjie Kong, Kailai Wang, Shipeng Wang, Xiojiao Wang, Xin Jiang, Yi Guo, Guojiang Shen, Xin Chen, and Qichao Ni. 2021. Real-Time Mask Identification for COVID-19: An Edge-Computing-Based Deep Learning Framework. IEEE Internet of Things Journal 8, 21 (2021), 15929–15938.

Nicholas Krichevsky, Matthew St Louis, and Tian Guo. 2021. Quantifying and improving performance of distributed deep learning with cloud storage. arXiv preprint arXiv:2108.06322 (2021).

Enghad Kristiani, Chao-Tung Yang, and Chiu-Yin Huang. 2020. iSEC: an optimized deep learning model for image classification on edge computing. IEEE Access 8 (2020), 27267–27276.

Aabhi Sheik Vijaya Kumar and Muthian Sivathanu. 2020. Quiver: An informed cache storage for deep learning. In 18th (USENIX) Conference on File and Storage Technologies (FAST) 20, 283–296.
and Health Informatics (2021).

187. Ogheeneuiri Oderiwoho, Hawzhin Mohammed, Tolouhe Odetola, Terry N Guo, Syed Hassan, and Felix Doghe. 2020. An Edge Intelligence Framework for Resource Constrained Community Area Network. In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 97–100.

188. Hideaki Oguni and Karuyuki Shudo. 2021. Communication Scheduling for 5G uplink in a dense indoor network. IEEE Access (2021).

189. Aniruddha Pant. 2021. Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare. In Medical Image Understanding and Analysis: 25th Annual Conference, MIAA 2021, Oxford, United Kingdom, July 12–14, 2021, Proceedings. Vol. 12722. Springer Nature, 451.

190. Dimitris Papadimitriou and Swayambhoo Jain. 2021. Data-Driven Low-Rank Network Compression. In 2021 IEEE International Conference on Image Processing (ICIP). IEEE, 3547–3551.

191. Eunhyeok Park, Donggyoung Kim, Soocheum Kim, Yong-Deok Kim, Gunhee Lee, Sungho Yoon, and Sungjo Yoo. 2015. Big little deep neural network for ultra low power inference. In 2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE, 124–132.

192. Jihong Park, Sumudu Samarawickrama, Mehdi Bennis, and Merozou Debahb. 2019. Wireless network intelligence at the edge. Proc. IEEE 107, 11 (2019), 2084–2239.

193. Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jihong Park, Sumudu Samarawickrama, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jihong Park, Sumudu Samarawickrama, and Frederico Lopes. 2021. Leveraging Edge Intelligence for Video Analytics in Autonomous Systems. In Proceedings of the 2nd Workshop on Collaborative Inference via Ensembles on the Edge. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 4690–4694.

194. Peyman Passban, Yimeng Wu, Mehdi Rezagholizadeh, and Qun Liu. 2020. ALP: Hierarchical early exits. arXiv preprint arXiv:2012.14022 (2020).

195. Fotini Patrona, Ioannis Mademlis, and Ioannis Pitas. 2021. Self-Supervised Clusters through Inter-Group Learning Methods in Healthcare. In IEEE Access (2021), 423–431.

196. Aniruddha Pant. 2021. Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare. In Medical Image Understanding and Analysis: 25th Annual Conference, MIAA 2021, Oxford, United Kingdom, July 12–14, 2021, Proceedings. Vol. 12722. Springer Nature, 451.

197. Ogheneuriri Oderhohwo, Hawzhin Mohammed, Tolouhe Odetola, Terry N Guo, Syed Hassan, and Felix Doghe. 2020. An Edge Intelligence Framework for Resource Constrained Community Area Network. In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 97–100.

198. Hideaki Oguni and Karuyuki Shudo. 2021. Communication Scheduling for 5G uplink in a dense indoor network. IEEE Access (2021).

199. Aniruddha Pant. 2021. Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare. In Medical Image Understanding and Analysis: 25th Annual Conference, MIAA 2021, Oxford, United Kingdom, July 12–14, 2021, Proceedings. Vol. 12722. Springer Nature, 451.

200. Dimitris Papadimitriou and Swayambhoo Jain. 2021. Data-Driven Low-Rank Network Compression. In 2021 IEEE International Conference on Image Processing (ICIP). IEEE, 3547–3551.

201. Eunhyeok Park, Donggyoung Kim, Soocheum Kim, Yong-Deok Kim, Gunhee Lee, Sungho Yoon, and Sungjo Yoo. 2015. Big little deep neural network for ultra low power inference. In 2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). IEEE, 124–132.

202. Jihong Park, Sumudu Samarawickrama, Mehdi Bennis, and Merozou Debahb. 2019. Wireless network intelligence at the edge. Proc. IEEE 107, 11 (2019), 2084–2239.

203. Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jihong Park, Sumudu Samarawickrama, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jihong Park, Sumudu Samarawickrama, and Frederico Lopes. 2021. Leveraging Edge Intelligence for Video Analytics in Autonomous Systems. In Proceedings of the 2nd Workshop on Collaborative Inference via Ensembles on the Edge. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 4690–4694.

204. Peyman Passban, Yimeng Wu, Mehdi Rezagholizadeh, and Qun Liu. 2020. ALP: Hierarchical early exits. arXiv preprint arXiv:2012.14022 (2020).

205. Fotini Patrona, Ioannis Mademlis, and Ioannis Pitas. 2021. Self-Supervised Clusters through Inter-Group Learning Methods in Healthcare. In IEEE Access (2021), 423–431.

206. Aniruddha Pant. 2021. Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare. In Medical Image Understanding and Analysis: 25th Annual Conference, MIAA 2021, Oxford, United Kingdom, July 12–14, 2021, Proceedings. Vol. 12722. Springer Nature, 451.

207. Ogheneuriri Oderhohwo, Hawzhin Mohammed, Tolouhe Odetola, Terry N Guo, Syed Hassan, and Felix Doghe. 2020. An Edge Intelligence Framework for Resource Constrained Community Area Network. In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 97–100.

208. Hideaki Oguni and Karuyuki Shudo. 2021. Communication Scheduling for 5G uplink in a dense indoor network. IEEE Access (2021).

209. Aniruddha Pant. 2021. Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare. In Medical Image Understanding and Analysis: 25th Annual Conference, MIAA 2021, Oxford, United Kingdom, July 12–14, 2021, Proceedings. Vol. 12722. Springer Nature, 451.
Ekanut Sothiwat, Liangli Zhen, Zengxiang Li, and Chi Zhang. 2021. Partially Encrypted Multi-Party Computation for Federated Learning. In 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 828–835.

Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggii. 2018. Sparsified SGD with memory. arXiv preprint arXiv:1809.07599 (2018).

Dimitris Stelidis, Hamza Salem, Tannay Ghai, Nikhil Dhingra, Umang Gupta, Chrysovalantis Anastasiou, Greg Ver Steeg, Seivastan Ravi, Muhammad Naqeed, Paul M Thompson, et al. 2021. Secure neuroimaging analysis using federated learning with homomorphic encryption. In 17th International Symposium on Medical Image Processing and Analysis, Vol. 12088. SPIE, 351–359.

Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In Sixteenth Annual Conference of the International Speech Communication Association.

Nour Alhuda Sulieman, Lorenzo Ricciardi Celsi, Wei Li, Albert Zomaya, and Lichao Sun. 2020. Splitfed: When federated learning meets split learning. 2469. IEEE, 2464–2470.

Máté Szabó. 2020. Distributed Machine Learning Using Data Parallelism on the Edge. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. 9825–9833.

Xinran Tan, Hongjia Li, Liming Wang, Xueqin Huang, and Zhen Xu. 2021. Empowering Adaptive Early-Exit Inference with Latency Awareness. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35.

Halvin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. 2018. Communication compression for decentralized training. Advances in Neural Information Processing Systems 31 (2018), 7652–7662.

Zhengheng Tang, Shashuai Shi, Xiaowen Chu, Wei Wang, and Bo Li. 2020. Communication-efficient distributed deep learning: A comprehensive survey. arXiv preprint arXiv:2003.06307 (2020).

Zhao Tao and Qiu Wen. 2018. Communication efficient distributed deep learning on the edge. In USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18).

Teerapattayon, Bradley McDaniel, and Haing-Tsung Kung. 2016. BrachyNet: Fast inference via early exiting from deep neural networks. In 2016 23rd International Conference on Pattern Recognition (ICPR). IEEE, 2464–2469.

Chandra Thapa, Mahawara Arachige Pathum Chamikara, Seiyt Camtepe, and Lichao Sun. 2020. SplitFed: When federated learning meets split learning. In Federated Learning Systems. Springer, 79–109.

Haibo Tian, Fangguo Zhang, Yufeng Shao, and Binghui Li. 2021. Secure Linear Aggregation Using Decentralized Threshold Additive Homomorphic Encryption For Federated Learning. arXiv preprint arXiv:2111.10753 (2021).

Nicola Tonellotto, Alberto Gotta, Franco Maria Nardini, Daniele Gadler, and Fabrizio Silvestri. 2021. Neural network quantization in federated learning at the edge. Information Sciences 575 (2021), 417–436.

Kentaroh Toyoda, Jun Zhao, Allan Neng Sheng Zhang, and P Takis Mathiopoulos. 2020. Blockchain-enabled federated learning with mechanism design. IEEE Access 8 (2020), 21974–219756.

Hong-Yen Tan and Daichun Hu. 2019. Privacy-preserving big data analytics a comprehensive analysis 3. Parallel and Distributed Comput. 134 (2019), 207–218.

Stacey Trues, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. 2020. LDP-Fed: Federated learning with local differential privacy. In Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking. 61–66.

Hideki Tsunashima, Hirokatsu Kataoka, Junji Yamato, Qiu Wen, and Shigeo Morishima. 2021. Adversarial Knowledge Distillation for a Compact Generator. In 2020 25th International Conference on Pattern Recognition (ICPR). IEEE, 10636–10643.

Mža Tziddin and Erik G Nilsson. 2020. Emotion recognition using speech and facial structural learning to facilitate edge intelligence. Engineering Applications of Artificial Intelligence 94 (2020), 103775.

Kota Anjani Vaibhavi. 2021. Survey on cryptographic algorithms in cloud computing. Journal of Research Proceedings 1, 2 (2021), 53–77.

Besson Variane, Nan Wang, David Bembich, Cheol-Ho Hong, Eyal de Lara, Weisong Shi, and Christopher Stewart. 2020. A Survey on Edge Performance Benchmarking. arXiv preprint arXiv:2004.11725 (2020).

Franzepa Vettempaka, Otkrist Gupta, Abhimmay Dubey, and Ramesh Raskar. 2019. Reducing leakage in distributed deep learning for sensitive health data. arXiv preprint arXiv:1912.00564 (2019).

Franzepa Vettempaka, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. 2020. NoPeek: Information leakage reduction to share activations in distributed deep learning. In 2020 International Conference on Data Mining Workshops (ICDMW). IEEE, 933–942.

John Violos, Titas Pagoulatou, Stylianos Tasakas, Konstantinos Tserpes, and Theodora Vorarjarige. 2021. Predicting Resource Usage in Edge Computing Infrastructures with CNN and a Hybrid Bayesian Particle Swarm Hyper-parameters Optimization Model. In Intelligent Computing: Springer, 562–580.

Isabel Wagner and David Eckhoff. 2018. Technical privacy metrics: a systematic survey. ACM Computing Surveys (CSUR) 51, 3 (2018), 1–38.

J. Wang, J. Wang, J. Wang, and J. Wang. 2018. Emerging paradigms of neural network pruning. arXiv preprint arXiv:2103.06460 (2021).

Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. 2018. Not just privacy: Improving performance of private deep learning in mobile cloud. In Proceedings of the 24th ACM SIGCSD International Conference on Knowledge Discovery & Data Mining. 2407–2416.

Kai Wang, Yu Liu, Qian Ma, and Quan Z Shen. 2021. MultiIDE: Multi-teacher Knowledge Distillation for Low-dimensional Knowledge Graph Embeddings. In Proceedings of the Web Conference 2021. 1716–1726.

Lipeng Wang, Qiong Luo, and Shengen Yan. 2021. DIES: Accelerating Distributed Deep Learning Tasks on Image Datasets. IEEE Transactions on Parallel and Distributed Systems (2021).

Shuai Wang, Yuncong Heng, Rui Wang, Qi Hao, Yik-Chung Wu, and Derrick Wing Kwan Ng. 2021. Edge Federating Learning Via Unit-Modulus Over-The-Air Communication (Extended Version). arXiv preprint arXiv:2101.12051 (2021).

Shibo Wang, Yunbo Li, Anwei Zhao, and Qiangmin Wang. 2021. Privacy Protection in Federated Learning Based on Differential Privacy and Mutual Information. In 2021 3rd International Conference on Artificial Intelligence and Advanced Manufacture. 428–435.

Xiaofei Wang, Yiwon Han, Victor CM Leung, Dusit Niyato, Xiaqing Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. IEEE Communications Surveys & Tutorials 22, 2 (2020), 869–904.

Xiaofei Wang, Yiwon Han, Chenyang Zhang, Qiang Zhao, Xu Chen, and Min Qin. 2019. In-Edge: Intelligentizing mobile edge computing, caching and communication by federated learning. IEEE Network 33, 5 (2019), 156–165.

Xiao Wang, Tianpeng Wei, Linghe Kong, Liang He, Fan Wu, and Guihai Chen. 2019. ECAS3: Edge computing based auxiliary sensing system for self-driving vehicles. Journal of Systems Architecture 97 (2019), 258–268.

Xiaoxi Wei, Pablo Ortega, and A Aldo Faisal. 2021. Inter-subject deep transfer learning for motor imagery eeg decoding. arXiv preprint arXiv:2103.03531 (2021).

Lahiru Welagedara, Janani Harischandra, and Nuwan Jayawardene. 2021. A Review on Edge Intelligence based Collaborative Learning Approaches. In 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 0572–0577.

Bin Wu. 2019. Hierarchical macro strategy model for moba game ai. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33. 1206–1213.

Huaming Wu, Ziru Zhang, Chuang Guan, Kai Xia, and Minxian Xu. 2020. Convergence of edge computing and distributed deep learning for smart city internet of things. IEEE Internet of Things Journal 7, 9 (2020), 8909–8911.

Yuncheng Wu, Shaofeng Cai, Xiaokai Xiao, Gang Chen, and Beng Ching Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. arXiv preprint arXiv:2008.06170 (2020).

Yong Xiao, Yingyu Li, Guangming Shi, and H Vincent Poor. 2020. Optimizing Resource-Efficiency for Federated Edge Intelligence in IoT Networks. In 2020 International Conference on Wireless Communications and Signal Processing (WCSPP). IEEE, 86–92.

Yong Xiao, Guangming Shi, Yingyu Li, Wilid Saad, and H Vincent Poor. 2020. Toward Self-Learning Edge Intelligence in 6G. IEEE Communications Magazine 58, 12 (2020), 34–40.

Ming Xie, Guodong Long, Tao Shen, Tianyi Zhou, Xinhai Wang, and Jing Jiang. 2020. Multi-center federated learning. arXiv preprint arXiv:2005.01026 (2020).
