Distributed and Collaborative High-Speed Inference Deep Learning for Mobile Edge with Topological Dependencies

Shagufta Henna, *Senior Member, IEEE,* Alan Davy, *Member, IEEE*

***Abstract*—Ubiquitous computing has potentials to harness the flexibility of distributed computing systems including cloud, edge, and internet of things devices. Mobile edge computing (MEC) benefits time-critical applications by providing low latency connections. However, most of the resource-constrained edge devices are not computationally feasible to host deep learning (DL) solutions. Further, these edge devices if deployed under denser deployments result in topological dependencies which if not taken into consideration adversely affect the MEC performance. To bring more intelligence to the edge under topological dependencies, compared to optimization heuristics, this work proposes a novel collaborative distributed DL approach. The proposed approach exploits topological dependencies of the edge using a resource-optimized graph neural network (GNN) version with an accelerated inference. By exploiting edge collaborative learning using stochastic gradient (SGD), the proposed approach called CGNN-edge ensures fast convergence and high accuracy. Collaborative learning of the deployed CGNN-edge incurs extra communication overhead and latency. To cope, this work proposes compressed collaborative learning based on momentum correction called cCGNN-edge with better scalability while preserving accuracy. Performance evaluation under IEEE 802.11ax-high-density wireless local area networks deployment demonstrates that both the schemes outperform cloud-based GNN inference in response time, satisfaction of latency requirements, and communication overhead.**

***Index Terms*—deep learning in edge computing, deep learning in cloud computing, edge inference, edge with topological dependencies, intelligent edge, intelligent cloud computing.**

I. Introduction

CLOUD-ASSISTED service provision for delay-sensitive applications cannot satisfy the latency constraints of time-sensitive applications where requests are served through the Internet backhaul. This remote cloud-assisted service provisioning also experiences uncertainties in latency, which can violate the latency requirements of delay-sensitive applications. with the advent of edge computing, delay-sensitive applications are set to witness actual realization, e.g., autonomous vehicles [1] and traffic management [2]. However, to meet the stringent latency requirements of these applications, the fast inference should be provided on the edge. Nevertheless, edge devices are resource-constrained with limited processing capability and memory.

Shagufta Henna is with the Telecommunication, Waterford Institute of Technology, Waterford, Ireland. e-mail: shaguftahenna@gmail.com

Alan Davy is with the Telecommunication Software & Systems Group, Waterford Institute of Technology, Waterford, Ireland. e-mail: adavy@tssg.org

Manuscript received July, 2019; revised August, 2019.

The availability of high power computational resources, e.g., GPUs have made deep learning-enabled solutions in high demand in diverse fields including computer vision, medical diagnostic, and networks. However, despite much progress on adapting these solutions, offloading latency sensitive application requests to remote clouds induces substantial communication overhead and latency, which is beyond the tolerance of time-critical applications.

Mobile edge computing is an emerging paradigm to circumvent longer delays by localizing the computation/processing to the close proximity of end devices [3], [4], [5]. The edge nodes, e.g., access points (APs) can serve the computation requests of latency-critical mobile and Internet of Things (IoT) applications. These edge nodes provide improved quality of service (QoS) without incurring the uncontrollable internet delay to remote cloud computing infrastructures. In MEC, the configuration of edge hosts should be adjusted dynamically according to the network conditions both in the short and long run [6].

Deep learning, as one of the popular paradigms of artificial intelligence, can be integrated into a mobile edge to cater to stringent application requirements by enabling fast inference. [7], [8]. A deep learning model is characterized by multiple layers, each with a varying number of neurons for feature extraction to predict the output with high accuracy. Due to high memory and computation requirements of these models, cloud-based deployments with sufficient resources are more conventional. Recent years have witnessed intelligent distributed computing systems [9], [10], where computing nodes dispersed in different geographic locations cooperatively provide ubiquitous computing optimizations driven by machine learning. These distributed computing systems target high availability and scalability under varying workloads. As these distributed computing systems have not yet calibrated deep learning-enabled inference on the edge. Therefore, these solutions incur higher communication overhead and are not suitable for time-critical applications.

In response to the excessive resource demand of deep neural networks, the traditional approach is to adopt a cloud datacenter for training. Data generated from end devices is sent to the cloud for processing, and then results are sent back to the end devices after inference. However, in this cloud-centric approach, data is uploaded to the remote cloud via a long wide-area network data transmission which can result in high end-to-end latency which is not desirable for time-critical applications. To alleviate the latency bottlenecks of the cloud-

centric approach, edge computing paradigm can be exploited to offload deep learning training and inference to the network edge. This offloading enables low-latency inference which is not affected due to the network transmission.

To cater the requirements of real-time applications, recently some optimization techniques have been proposed to deploy the trained deep learning models on resource-constrained edge devices, e.g., fixed-point quantization [11]], networking pruning [12], and hardware/software acceleration [13], [14], etc. However, most of these deep learning schemes are applicable to scenarios, where end devices are not always connected to the network. In addition, recently transmission scheduling schemes based on shared deep neural network (DNN) models [15] have been proposed. These schemes exploit data parallelism to scale-down the size of data considering the resource constraints of edge devices. Although, DNN-based inference on the edge provides accurate and reliable inference, however, these applications require extensive computation resources, which most of the edge devices cannot well support. Even deployed on the edge, DNN-based inference requires higher processing due to a large number of computationally-extensive layers. Considering mission-critical applications,e.g., AR/VR games, a natural research challenge is to further improve the latency of edge-based DNN inference.

The proposed idea attempts data parallelism in the context of deep learning like federated learning [16].However, federated learning still relies on a central parameter server. The proposed solution is suitable for situations without a central server and is also robust under central parameter server failures.

In the context of IEEE 802.11ax, dense deployment of APs can result in inter-WLAN interactions. Under these deployments, partially overlapping APs play a vital role in creating spatial dependencies called topological dependencies. Under these scenarios, a configuration of an AP can negatively impact the performance of neighboring APs. Precisely, these dependencies can affect the potential of a WLAN to exploit spatial reuse, thereby limiting its average network throughput.

Conventionally, the above discussed edge-based deep learning schemes consider the computation and memory resources of edge devices and pay no consideration to the topological dependencies that exist between the edge devices within an existing deployment. These dependencies have a significant impact on the performance of the overall network, where the configuration of an edge device affects the performance of its neighboring edge devices. How to achieve cost-efficient fast inference under topological dependencies is crucial to mobile edge operators when considering dense edge device deployments.

To address the above question, this work adopts the GNN model on edge devices [17]. GNN derives the topology information of each edge device from combining its past configuration settings as well as the modeled effect on the same from edge devices in its neighborhood. Specifically, this paper proposes a novel distributed collaborative deep learning solutions for fast inference called GNN-edge. On one hand, the proposed schemes exploit the GNN approach to capture the topological dependencies of the edge devices to optimize configuration under dynamic and dense scenarios. On the other hand, the proposed schemes partition the GNN to fit more realistic edge devices with capacity constraints. The early-branching mechanism of the proposed schemes assures fast inference to meet the latency requirements of time-critical applications. To ensure the accuracy and reliability of the GNN-edge, this work proposes collaborative learning on the edge by adopting collaborative SGD and compressed collaborative SGD. In specific, the optimal configuration of edge device is inferred with the help of topological dependencies coupled with the partitioning and early-branching of branchynet which is further augmented by the collaborative learning among edge devices to meet best the requirements of time-critical applications.

The main contributions of this work are summarized as follows.

- The GNN approach is used to exploit the topological dependencies of the resource-constrained edge devices for efficient inference. Considering resource constraints of edge devices and application requirements, GNN is partitioned to keep the computationally less intensive layers on the edge with an early-branching mechanism based on the latency requirements called GNN-edge.
- By leveraging the benefits of local data, CGNN-edge is trained online by using collaborative learning on the edge called as collaborative GNN-edge (CGNN-edge).
- To reduce the communication overhead incurred by the collaborative learning by the edge devices, GNN-edge is trained by using compressed collaborative learning algorithm called as compressed collaborative GNN-edge (cCGNN-edge).
- The use case of indoor IEEE 802.11ax-based dense WLAN deployment is modeled by using GNN. In an indoor WLAN dense deployment, topological dependencies between APs are more frequent and affect the overall network performance in terms of throughput.
- To evaluate the performance of CGNN-edge and cCGNN-edge algorithms, simulations based on the use case of indoor IEEE 802.11ax-based dense wireless local area networks (WLANs) deployments are implemented. Results demonstrate that the CGNN-edge and cCGNN-edge outperform GNN-based inference on the cloud called GNN-cloud in terms of inference speed and communication-overhead with reasonable accuracy.

This paper is organized as follows. Section II discusses the related work. In Section III, the use case for the IEEE 802.11ax-based WLAN under dense deployment is presented. Section IV provides the details of the system model used for the proposed schemes. In Section V, spatial reuse in IEEE 802.11ax WLANs is modeled by using GNN. Section VI presents the proposed collaborative graph neural network approach for the edge, i.e., CGNN-edge. Section VII describes the proposed compressed collaborative GNN-based learning cCGNN-edge, and evaluations along with analysis are presented in Section VIII. Finally, the conclusion and future work are given in Section IX.

II. Related Work

Compared to the conventional cloud computing, the new era of edge computing has demonstrated significant benefits in terms of memory cost, energy consumption, and low latency for a broad range of big data applications.

Khelifi et al. investigated the applications of deep learning models, e.g., convolutional neural networks (CNNs), recurrent neural networks (RNNs), and reinforcement learning (RL) in edge computing. Authors in [19], presented an offloading strategy to deploy deep learning models to optimize the performance of edge computing. In some of the works [20], [21], gradient-descent-based distributed solutions to machine learning have been investigated theoretically in terms of training convergence and communication overhead. These works further have been optimized to general communication-efficient distributed frameworks, e.g., [22]. Although, these schemes demonstrate a reduction in communication overhead. However, none of them consider scenarios with topological dependencies among the edge devices which affect the overall network performance.

An offload deep learning scheduling scheme for the edge has been proposed in [23] to optimize the network performance. In this scheme, the deep learning tasks are deployed both at the edge devices and edge servers. However, the proposed scheme relies on the known service capacity and bandwidth availability and do not consider the dynamics of the edge layer and its effect on network performance. Recent works on deep reinforcement learning for the MEC has been reported in [24], [25]. These works consider the use of deep Q-networks to optimize MEC. These solutions, however, suffer from the drawbacks of Q-table entries which are not scalable for high dimensional space.

According to works in [26] and [27], the deployment of a trained deep learning model on resource-constrained edge devices is achievable with network pruning, quantization, and Huffman encoding. This and most of the other works rely on pruning and quantization schemes to work on the resource-constrained edge. To optimize the operation of the MEC, these schemes are coupled with special hardware and software accelerators [28], [29]. Authors in [30] proposed a CNN-based network architecture to improve the energy efficiency of IoT devices. The proposed architecture aims to improve deep learning accuracy on resource-constrained edge devices while still striving to cope with the real-time latency constraints of surveillance applications.

An advanced deep reinforcement learning approach deep Q-network (DQN) for computation offloading on multiple base stations is proposed in [31]. The DQN-based offloading algorithms learn an optimal policy to improve the long-term network utility. More specifically, DQN-based reinforcement learning utilizes a deep neural network instead of a Q-table. Different compression techniques to reduce the complexity of neural networks have been proposed in recent literature [32], [33]. These schemes aim to meet the requirements of resource-constrained devices. However, these schemes suffer in accuracy.

There is some recent work on the deployment of deep neural networks on mobile devices within a WLAN [34]. The work proposed a distributed programming model among mobile devices to accelerate CNN inference. The basic approach is to exploit model parallelism by partitioning the individual layers into slices to fit them on memory-constrained mobile devices. Li et al. [35] introduced a framework for adaptive mobile object recognition. It consists of an edge master server to control and train the mobile devices based on domain-aware adaptation model. Teerapittayanon et al. [36] presented a distributed deep neural network approach, which partition the deep neural network across the cloud, edge, and mobile devices to reduce latency. Although, the proposed approach achieves better latency, however, it provides no mechanism to consider edge dependencies coupled with collaborative learning.

The authors in [49] also concern offloading in MEC, they investigate the application of machine learning-based approaches, which perform better than traditional approaches. Based on their investigation, the authors proposed a preliminary design of an augmented intelligence-based MEC system. The proposed framework is expected to construct a smart MEC decision system based on the principles of artificial intelligence. In [50], a multi-stage stochastic programming model for a large-scale cooperative crowd-sensing system is proposed. The system is based on an incentive-based mechanism for distributed resource allocation coupled with multi-stage stochastic programming. It also offers an auction-based migration for load-balancing. The former work proposes a preliminary smart MEC decision system while the later discusses the design of a game-theoretic incentive mechanism for resource allocation with no consideration to collaborative learning and topological dependencies.

Although there is a strong drive towards edge-based inference, however, still most of the deep learning applications, e.g., speech recognition are cloud-assisted. In a work [37], authors introduced a novel deep learning model which can be deployed and executed on wearable small IoT devices to assist tasks driven by audio recognition. Authors in [38] introduced a framework based on deep learning and Apache spark to perform IoT data analytics. It consists of two layers. The first layer is the deployment of inference on mobile devices. On the other hand, the second layer consisting of Apache Spark is deployed on the cloud to train the neural network. This framework reinforces the possibility of offloading some of the processing to the edge. Liu e al. [39] proposed the first work on deep learning-based edge intelligence. Primarily, authors used the edge intelligence infrastructure for food recognition applications with an aim to reduce response time and energy consumption. This work considered mobile devices as edge devices. However, mobile devices do not reflect the realistic edge of the network and do not have better support for executing full-fledged deep learning algorithms.

With the advent of edge computing, prior work has studied the limitations of edge devices with a focus on lightweight learning models on the edge [40]. However, compared to the prior work which focuses more on model choices and deployment decisions, our work considers inference on the edge while considering topological dependencies. We propose

local inference for more complex edge scenarios. Another work in [41] considers splitting of deep neural network layers among the cloud and edge to reduce energy consumption and latency, however, the works focuses only on image processing applications and does not address the real-interactions among the edge devices. In another work [16], the authors proposed a federated learning concept which exploits a group of smartphones to train the model in an online manner. Each device periodically updates the locally trained model to the cloud which computes and communicates back the updated average weight. In contrast, our work proposes a local collaborative training process which requires no assistance of the cloud to update the weighted gradient average on all the edge devices.

Overall, our work differs from the above in that we investigate deep learning algorithm and deployment which consider fast inference assisted by local collaborative training in which topological edge dependencies are exploited to improve the overall model prediction performance, speed, and accuracy. To our knowledge, this is the first approach which captures the edge topological dependencies with a more realistic edge-compatible GNN approach with faster inference assisted by the collaborative learning.

III. IEEE 802.11 WLANs USE CASE

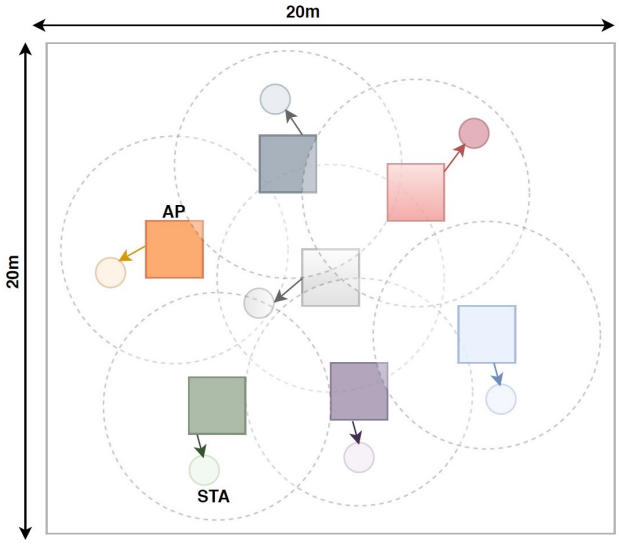

Fig. 1. High density WLAN deployment.

Future wireless networks require massive connectivity which will be heavily dependent on the high-density indoor and outdoor deployment of access points. A highly-dense WLAN deployment scenario represents a large number of APs deployed within a fixed region with a purpose to provide massive connectivity. Few examples of typical denser scenarios that IEEE 802.11 ax needs to support include enterprise office scenario, outdoor large hotspots, dense residential apartments, and stadiums. The main objective of IEEE 802.11ax dense deployment is enhanced user experience with at least 4 times throughput improvement for each station. As a challenge, this dense deployment of APs often results in partially overlapping scenarios with coexistence issues. This requires the IEEE 802.11ax deployment to meet the following requirements:

- Efficient use of spectrum under densely deployed APs and stations
- An improved performance in terms of average network throughput
- An efficient interference management technique under coexisting APs

Figure 1 shows an example of next-generation high-density WLAN deployment where multiple WLANs coexist. In such scenarios, partially overlapping APs can negatively impact the performance of neighboring WLANs. If not controlled appropriately, this can further aggravate the performance of IEEE 802.11ax WLANs with the exposed and hidden terminal problem, high contention, flow starvation, and other network asymmetries.

IV. SYSTEM MODEL

A. Deep Learning-enabled Mobile Edge Architecture

In our proposed deep learning-enabled mobile edge architecture, IEEE 802.11 ax-enabled APs are assumed to be equipped with agents which can monitor the collected information. These agents can modify the configurations of APs to optimize network performance, e.g., an agent can select a channel or change transmission power. This information can also be used by different learning algorithms in a centralized, decentralized, and distributed manner to predict an optimal AP configuration. In the decentralized mode of learning, agents exploit local information to select the configuration of a WLAN. Agents in this mode require communication with the AP with negligible communication latency. Decentralized algorithms, e.g., Carrier Sense Multiple Access with Collision Avoidance used in 802.11 WLANs under uncoordinated deployments suffer in terms of throughput and scalability. On the other hand, in distributed learning mode, agents decide a global configuration that can be applied to a dense deployment of APs. In this mode, agents need to rely on a collaborative strategy to acquire additional information from the environment. This information can be used by an agent to decide its optimal configuration [42].

In addition to the collection, 802.11 ax-enabled agents can also exchange the collected information with the other entities, e.g., neighboring agents or a central controller (CC) deployed on the cloud. The deep learning-enabled edge architecture considered this work is based on [43]. It consists of two major layers: the edge layer and could layer as illustrated in Figure 2. The APS equipped with the agents forms the edge layer, whereas, CC represents the cloud layer. The architecture is summarized below:

1) Access Point: APs represent the edge of the network for this work and their operation is controlled by the deep learning-driven agent as discussed in the next section. APs are integrated with the agents with the help of two modules: monitoring and communication. The monitoring module collects the information of wireless operation and supplies it to the agent through the communication module. The collected information includes performance statistics, network, and configuration

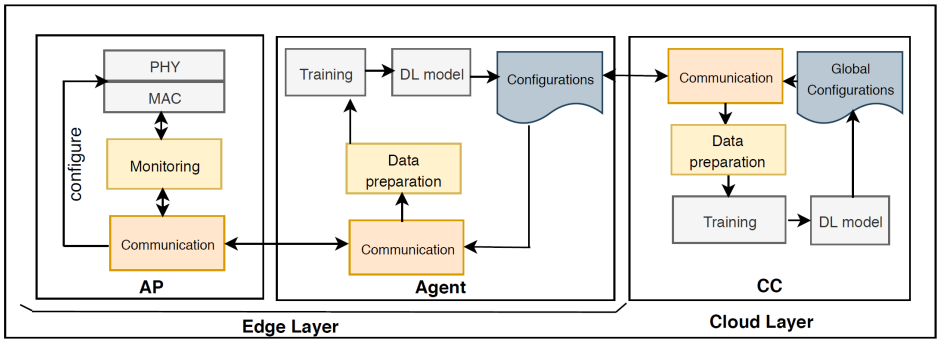

Fig. 2. Deep learning-enabled mobile edge architecture.

information. The communication module is also responsible to communicate predictions made by the agents to the AP to modify its configurations synchronously or asynchronously.

2) Agent: An agent handles the data received from the AP and utilizes it for the training/retraining of deep learning module deployed either locally in a distributed manner or at the CC on the cloud. Similar to AP, it consists of a communication module, data preparation and training module, and a deep learning algorithm. Communication module ensures information exchange between AP to agent, agent to agent, and agent to CC. Our proposed distributed collaborative learning mechanism relies on agent-to-agent communication. On the contrary, the centralized training mechanism implemented by the CC requires agent-to-CC communication with no effect on an agent ś deep learning module. Interactions from agent-to-agent and agent-to-CC assume an underlying IEEE 802.11k communication mechanism. Data preparation module performs the necessary pre-processing to transform the data in a form which can be used to train the deep learning module. Training module updates the deep learning model based on the data received from the data preparation module. Finally, the trained deep learning model predicts the new configuration of the AP and communicates it to AP through the communication module.

3) Central Controller (CC): The CC deployed on the cloud controls and coordinates all the agents synchronously and asynchronously. Similar to an agent, it consists of communication, data preparation, training, and deep learning modules. The communication module is responsible to exchange information with the agents. Agents collect and report the information to the CC, whereas, CC communicates the predicted configuration of the deep learning module to the agents. Any preprocessing on the collected data/available data is performed by the data preparation module. The CC can train/retrain the deep learning model based on the pre-processed data. It schedules retraining based on the surge reported by the agents. As sufficient computation resources are available on the CC, it is possible to deploy more powerful deep learning modules at the CC for accurate predictions.

B. Distributed learning

Similar to Komondor [43], the agents in the edge layer collaborate to predict the configuration of WLANs by exchanging information and implementing the DL model in a distributed manner. Additional information collected by the agents is used for collaborative learning as discussed later. Distributed learning among agents is illustrated in Figure 3 where agents can exchange their configuration information.

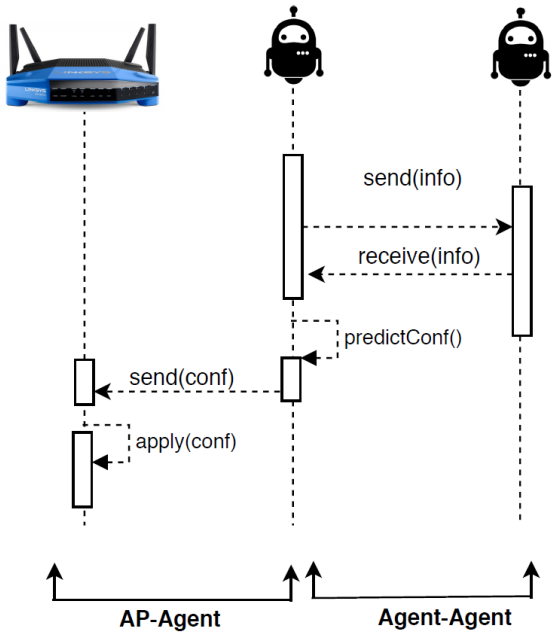

Fig. 3. Agent-agent communication in the distributed learning model.

C. Graph Neural Network Model

Our work adopts Graph Neural Network (GNN)-based [17] approach to model the topological dependencies of the edge layer from Figure 2. These dependencies impact the performance of APs in terms of efficient spectrum utilization, fairness, and throughput. Given an indoor high-density WLAN deployment as shown in 1, the use of GNN is motivated by

the fact that a denser WLAN deployment results in inter-AP dependencies concerning the channel, transmit power and carrier sensing power.

Similar to [17], GNN is assumed with two feed-forward neural networks (FNN) with different types of layers, e.g., convolution and pooling. The deep learning module deployed on both the agent and CC is based on GNN. Given a trained GNN, an agent can predict the configuration based on its historical features as well as the features of neighbouring agents. CC is assumed to be provided with the topological dependencies in a WLAN deployment through agents. An AP utilizes the first FNN of GNN to compute its next state s_n based on the features of its neighbouring AP denoted as n^* , whereas, the second FNN predicts the new configuration based on the new state and its historical features. This can be explained with the help of Equation 1 and Equation 2 as given below.

$$s_n = \sum_{m \in n} h_w(f_n, f_m, s_m), \forall n \tag{1}$$

$$O_n(t) = g_w(s_n, f_n), \forall n \tag{2}$$

In the Equation 1 , f_m and s_m denote the features and states of neighbour m. h_w is a parametric function and represents the dependence of state of each node on the states of its neighbors. g_w represents a node ś dependence on its next state and features. In GNN, a graph is processed by using both the h_w and g_w units corresponding to each node. The diffusion process of these units tends to converge exponentially fast with stable node states and predicted output.

V. Graph Neural Network Model for the Spatial Reuse

Given a coexisting WLANs scenario, the spatial reuse aims to improve spectral efficiency. Although, several spatial reuse mechanisms are possible, however, this work focuses on the prediction of sensitivity and transmission power configurations in a distributed and collaborative manner.

The transmission and sensing power fluctuations, channels, and sensitivity threshold at one AP can adversely affect the performance of neighboring APs, thereby reflecting critical topological dependencies. These topological dependencies can be best modeled with GNN for faster and accurate configuration predictions which can significantly improve the spectral efficiency. Throughout the rest of the paper, the inter-dependencies among APs as inter-dependencies among agents means the same.

GNN-based spatial reuse for the high-density WLANs is primarily based on the use of Equation 1 and Equation2. An example of inter-dependencies among agents is depicted in Figure 4. The components involved in GNN-based configuration prediction are illustrated in Figure 5 and are discussed below.

A. WLAN-AP Features

The WLAN-AP features are the observations or monitored information from the neighboring APs or Agents. All WLAN-AP features including the agent itself and its neighboring agent are provided as an input to the state function in Equation 1. However, the output function as given in Equation 2 requires only the features of the agent on which it is deployed. In the context of WLAN, these features include channel information, transmission and sensitivity power, throughput, fairness, and bit error rate (BER), etc.

B. WLAN-AP States

This work assumes that the APs are stateful, where at any instant of time an agent has a state. The state s_n of each agent can be derived by the parametric function h_w which is based on an agent ś own features and the features of its neighboring agents. It can be inferred therefore that the state of an agent reflects the topological dependencies of the WLAN.

Figure 4 shows the inter-dependencies of agents under a deployment of 5 WLANs. The figure depicts that, for example, the state of $agent_1$ depends on the states of four directly connected agents, and their corresponding features, i.e., f_2, f_3, f_4, f_5. Based on Equation 1, state of $agent_3$, denoted as s_3 in the Figure 4 can be given as Equation 3.

$$s_3 = h_w(f_3, f_2, s_2) + h_w(f_3, f_1, s_1) \tag{3}$$

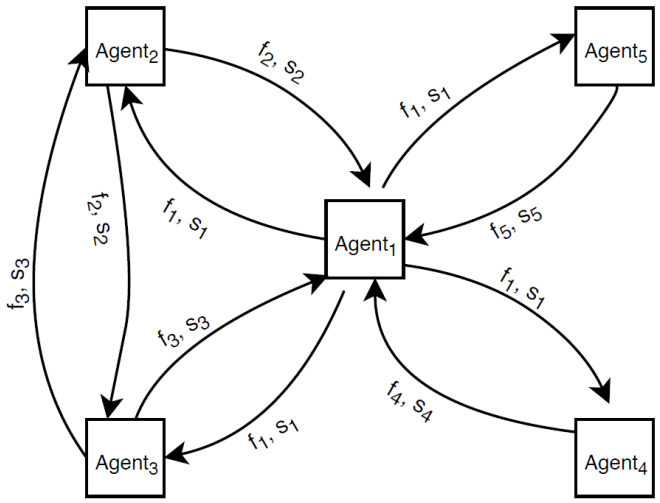

Fig. 4. States & Features from an Agent ś neighbourhood.

C. Output Function

The predicted configuration of an agent depends on the next state computed by the h_w and its own features. The output function for the example of 4 is given in Equation 4, where g_w function is an FNN which can be trained by an appropriate gradient calculation. For our proposed work, this gradient is computed in a distributed and collaborative manner. The g_w function of each agent predicts the transmission and sensitivity power to be used for the next round.

$$O_3(t) = g_w(s_3, f_3) \tag{4}$$

VI. Collaborative Graph Neural Network Approach for the Edge (CGNN-edge)

A. Overview

Topological dependencies in denser WLAN deployments are one of the striking factors which can affect the performance of WLAN in terms of throughput, fairness, and spectrum efficiency. This section proposes a collaborative enhancement to GNN called as CGNN-edge to predict the configurations of an AP locally and dynamically. Since neighboring APs in an indoor WLAN deployment likely coexist, an overlapped AP is expected to influence the performance of its neighboring APs. This dependency of AP on their neighborhood makes the GNN model an interesting fit to predict configurations in a denser indoor WLAN deployment. GNN deployment and local processing on APs can climb up high incurring higher latency which is not acceptable to mission-critical applications, e.g., virtual reality/augmented reality games and smart factories. To address these issues, on one hand, CGNN-edge partitions the GNN and deploys the computationally less intensive layers on the APs. On the other hand, it also accelerates the GNN interference by branching early based on application requirements and constraints. CGNN-edge deployment considers only deep learning agents involved in the edge without any coordination to a central parameter server. Specifically, this work proposes a collaborative deep learning mechanism to train the accelerated GNN model on the edge. The frequent communication among the agents to synchronize gradients incurs extra network communication overhead. To overcome, this work also considers compressed gradient-based optimization to CGNN-edge called as cCGNN-edge which synchronizes only important gradient coordinates.

The next configuration predicted by the CGNN-edge or cCGNN-edge is presented in Algorithm 1, which in turn uses the Algorithm 2 to extract the GNN layers based on the branchynet model [18]. Collaborative training of CGNN-edge is given in Algorithm 3, whereas a compressed collaborative training version for the CGNN-edge is illustrated in Algorithm 4.

B. Description

CGNN-edge-enabled configuration prediction is primarily defined by the Equation 1 and Equation 2 and is illustrated in Algorithm 1. It is further optimized adaptively by the GNN partition to meet the constraints of edge devices as given in Algorithm 2. To accelerate the CGNN-edge inference, the Algorithm 2 branch at an earlier GNN layer. This significantly reduces the computation latency to better address the requirements of time-critical applications including autonomous cars and smart factories etc.

The algorithm 1 takes as an input the AP profile which can be used to partition the GNN by the Algorithm 2. Within Algorithm 2, line 1 observes and collects the features of a WLAN deployment, i.e., features of an AP and its neighborhood. These features include channel, transmission power (T_x), and sensitivity levels (S). It can be observed from line 2 of the Algorithm that GNN is partitioned coupled with the early branch points according to the AP profile. To achieve this, Algorithm 2 is called with the AP profile which returns the optimal number of GNN layers to proceed. As can be seen within Algorithm 1, lines 6-7 computes the next state of the AP based on the h_w function. This is an iterative process to compute the state of each AP involving as many as h_w functions for AP as its neighboring APs in the WLAN deployment. Within Algorithm 1, lines 10-11 determine the output state for the configuration prediction based on the states as computed in the lines 5-8.

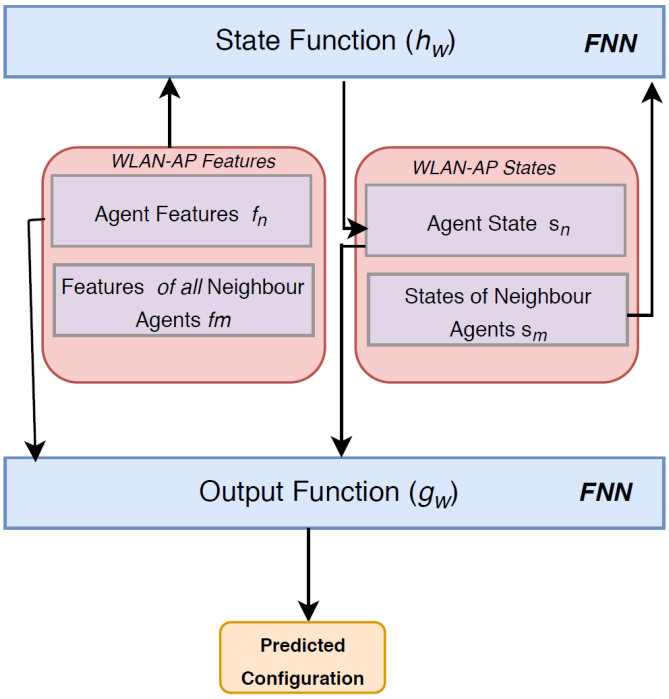

Fig. 5. GNN-based prediction of configuration for a single AP/Agent.

The algorithm 2 mitigates the performance bottleneck of CGNN-edge execution. As the CGNN-edge layers exhibit different runtime, therefore, lines 3-6 of the Algorithm 2 partition the GNN into two parts, i.e., EAP and SP. EAP represents computationally less intensive layers which can be deployed on the edge. On the other hand, computationally intensive layers SP are offloaded to a computationally sufficient server. To calibrate the low-latency inference on the edge, Algorithm 2 only focuses on computationally less intensive layers deployed on the edge as illustrated in lines 8-11. The CGNN-edge is further optimized for fast inference by exploiting the benefits of branching, where a trained CGNN-edge can branch earlier based on the latency constraints of an application. Algorithm 2 uses the BranchyNet [18] model to train the CGNN-edge with multiple branches as presented in lines 1-6 of the Algorithm 2. Algorithm 2 utilizes the off-line training to profile the APs and regression models for each CGNN-edge layer, i.e., $R(L_{GNN})$ to predict the performance of each CGNN-edge layer, which is followed by the Branchynet-enabled training with multiple branches to accelerate the inference.

As illustrated in the lines 7-20 of the Algorithm 2 that based on the application requirements, the CGNN-edge is optimized online using early branching. This optimized CGNN-edge

model is deployed and executed on the access points. We use the same regression models as given in [eddgent] to predict EAP_i, i.e., the runtime of the j^{th} layer on the AP. These regression models for each layer are trained offline and therefore can be used directly for the performance prediction of each layer, thereby accelerating the CGNN-edge inference on the edge. An early branching of CGNN-edge, although accelerates the inference, however, it brings along the accuracy constraints.

To train the CGNN-edge, the proposed collaborative learning mechanism is presented in the Algorithm 3. The proposed collaborative learning mechanism is based on consensus-based distributed SGD (CDSGD)[fixedtopo]. However, unlike CDSGD, the proposed learning mechanism utilizes local data to train the CGNN-edge collaboratively among neighboring APs. It couples the data parallelism with the local data along with the neighborhood information to update the CGNN-edge.

In Algorithm 3, initially, a mini-batch of data is distributed among the agents thus exhibiting data parallelism. However, to make sure that the CGNN-edge learning takes into account the network dynamics, each agent keeps track of its local data by using the monitoring module as discussed in the Section IV and illustrated in line 2 of the Algorithm 3. The collected data is appended with the mini-batch to make an agents' respective training dataset. Further, as our work focuses more on topological dependencies, therefore, the proposed collaborative learning algorithm assumes that the agents can communicate with each other as explained in the distributed learning in Section IV.

The Algorithm 3 is provided with the D_q, where $q = 1 \dots N$. It denotes the mini-batch of the training data with u_q samples corresponding to the q agent. Within the Algorithm 3, lines 5-7 are based on the concept of SGD and consensus, where the $N(q)$ denotes the neighborhood of q agent. The α represents the step size and $g_x(w_q^p)$ is the stochastic gradient for the local training data set w_q at epoch p^{th}. Specifically, the $g_x(w_q^p)$ depends on the size of the local training data set w_q. Further details of the CDSGD versions based on Polyak momentum and Nesterov momentum can be referred from the [44], [45].

VII. Compressed Collaborative Learning for the CGNN-edge (cCGNN-edge)

The collaborative training of the CGNN-edge suggests that the training gradients are sparse where most weights are close to 0. This is mainly attributed to the small training samples available on the APs. These gradients can be represented as multi-dimensional vectors where individual elements of gradients are called gradient coordinates. The frequent exchange of these small gradient coordinates can result in high communication overhead among APs. The sparsity of these gradients suggests that only a small fraction of these gradient coordinates need to exchanged among neighboring APs after update of each local data set. Simply dropping small value gradient coordinates can affect the training accuracy of the CGNN-edge. Unlike these gradient drop approaches, this work adopts the communication-efficient gradient update approach

Algorithm 1: AP configuration based on CGNN-edge

Data: $AP_{profile}$, $k = 0$
Result: Predicted output configuration using C
1 Collect features for all APs and their neighbourhood
2 CGNN-edge $\Leftarrow$ Algorithm2($AP_{profile}$)
3 ▷ Extract CGNN-edge
4 $M \Leftarrow$ Layers $(G(N, E)$ ▷ Get layers of CGNN-edge
5 **while** $k < M$ **do**
6 compute next state $s(k+1)$ using $s_n(k+1) = \sum_{m \in n} h_w(f_n, f_m, s_m(k)), \forall n$
7 $k \Leftarrow k + 1$
8 **end**
9 ▷ Predict configuration
10 $O_k = g_w(s_n(k), f_n), \forall n$
11 $C \Leftarrow O_k$
12 return C

Algorithm 2: Graph neural network for the edge

Data: Exit points E in branchy model, N Number of layers in each exit point E, regression models for each layer $R(L_{GNN})$
Result: CGNN-edge
1 **while** $E \neq 1$ **do**
2 $i \Leftarrow E$
3 **for** $j = 1 \dots N_i$ **do**
4 $EAP_j \Leftarrow R_{AP}(L_{GNN_j})$
5 $SP_j \Leftarrow R_S(L_{GNN_j})$
6 **end**
7 $profile(GNN_{i,p}) = \underset{p=1\dots N_i}{argmin}(\sum_{j=1}^{p-1} EAP_j + \sum_{y=p}^{N_i} SP_j)$
8 **if** $profile(GNN_{i,p})) = AP_{profile}$ **then**
9 CGNN-edge $\Leftarrow GNN_{i,p}$
10 return CGNN-edge
11 **end**
12 **end**

Algorithm 3: Collaborative training of CGNN-edge

Data: m, α, N,w_0^q, $qx = 1, 2, \dots N$
1 ▷ batch of local data on an agent q
Result: Collaborative training of $CGNN_{edge}$
2 Each AP/agent x keeps track of batch of its local trainable data
3 $W \Leftarrow 0$
4 **for** *Each Agent* x **do**
5 **for** $p = 0 \rightarrow m$ **do**
6 $z_q^{p+1} \Leftarrow \sum_{l \in N(q)} \pi_{ql} w_p^l$
$w_q^{p+1} \Leftarrow z_q^{p+1} - \alpha g_x(w_q^{p+1})$
7 **end**
8 **end**

as given [46]. The eSGD approach targets communication-efficient gradient updates between the edge and the cloud. However, unlike eSGD, the proposed training approach exchange and synchronize gradient updates among agents with topological dependencies to achieve communication-efficient collaborative online training.

Instead of dropping all the least significant gradient coordinates, similar to [46], the Algorithm 4 selects a set of critical gradient coordinates which should be exchanged. These critical gradient coordinates contribute effectively to collaborative learning to converge faster without any significant loss of accuracy. To determine the critical gradients which need to be synchronized, each gradient coordinate is assigned a weight which is a positive value. Each time a gradient coordinate contributes to the collaborative leaving of CGNN-edge, its weight is updated with a positive value. A gradient coordinate with a larger weight is considered as the critical gradient to be synchronized and exchanged next time.

Within the Algorithm 4, in lines 3-4, each agent/AP updates the weights of gradient coordinates and loss at time t. It also updates threshold $cCGNN_{thr}$ according to the current gradient average as illustrated on line 5 to exchange the accumulated gradients as explained later. Similar to other SGD methods, the Algorithm 4 strives to minimize the loss function C for the at any instant of time as given on line 7-9. If the loss at time t is less than $t-1$, it records the gradient coordinate index denoted as $g_{i,t}^{edge}$ and updates its weight represented as $F_{i,t}^{x}$. On the other hand for a higher loss function, line 11 of Algorithm 4 executes random weight selection to assign weights to gradient coordinates. The random weight selection procedure ensures that in the next round, gradient coordinates with large weight will be more likely selected to be exchanged by the agents. Algorithm 4 strives to keep loss function at t less than $t-1$ to converge to an optimal loss function.

To ensure less harm to accuracy and convergence, line 13 of Algorithm 4 also considers the small gradient coordinate values. It accumulates these residual values and applies momentum correction on it similar to deep compression [47]. To avoid the delayed outdated residual values, the sum function on line 13 considers a discount factor β which update the small but biased gradient coordinate values as given in Equation 5.

$$g_{i,t}^{sum} = \beta g_{i,t-1}^{sum} + (1-\beta) g_{i,t} \tag{5}$$

Line 13 within the Algorithm 4 illustrates that as the $g_{i,t}^{sum}$ exceeds the predefined threshold, i.e., $cCGNN_{thr}$, agents/APs synchronize them with their neighbouring agents/APs as given in Line 20. Line 15 of the Algorithm shows that the next gradient coordinate to be exchanged $g_{i,t}^{edge}$ is replaced with the $g_{i,t}^{sum}$ after discarding the least significant weighted coordinates.

Upon receiving the gradient coordinates from the neighbors, the agent as given in Lines 21-13 of the Algorithm 4 updates its gradient coordinate g_x^t according to the average gradient coordinates of its neighbors to train the CGNN-edge called now as compressed CGNN-edge (cCGNN-edge).

Algorithm 4: Compressed collaborative GNN learning (cCGNN-edge) on AP x

Data: m, α, N,F, $qx = 1, 2, \ldots N$)
1 ▷ batch of local data on an agent q
Result: Collaborative training of $CGNN - edge$
2 $F_0^x \Leftarrow 0$
3 Each AP/agent x updates gradient coordinate weights at time t
4 $C_t \Leftarrow C_{t-1} - g_x^{edge}$
5 $cCGNN_{thr} \Leftarrow \sum_I \frac{g_{i,t-1}}{I}$
6 **if** $(t > 1)$ **then**
7 **if** $(loss(C_{t-1}) > loss(C_t))$ **then**
8 $g_{i,t}^{edge} \Leftarrow g_{i,t-1}^{edge}$
9 $F_{i,t}^{x} \Leftarrow F_{i,t-1}^{x}$
10 **else**
11 $g_t^{edge} \Leftarrow rand(weighted(g_{t-1}^{edge}, F_{i,t}^{x})), \forall i$
12 **end**
13 $g_t^{sum} \Leftarrow momemtum_{correction}(Sum(g_{t-1}^{sum}))$
14 **while** $(g_t^{sum} > cCGNN_{thr})$ **do**
15 $g_{i,t}^{edge} \Leftarrow droplsig(g_t^{sum})$
16 **end**
17 **else**
18 $g_x^{edge} \Leftarrow rand(g_t^{sum})$
19 **end**
20 send g_x^{edge} to N_x
21 **if** $(Received(g_x^{edge}))$ **then**
22 $g_{x,t}^{edge} \Leftarrow average(g_t^{edge}, N)$ ▷ Update $g_{x,t}$ according to average gradient of neighbours
23 **end**

VIII. System Evaluation

This section evaluates the proposed CGNN-edge and cCGNN-edge within an indoor WLAN dense deployment based on komondor [43]. The evaluations demonstrate various advantages of the proposed techniques in terms of better throughput, lower configuration time, and low communication latency. In this section, the evaluations of proposed schemes focus on spatial reuse, i.e., to control the sensitivity and transmit power to reduce interference. However, the proposed schemes can be applied for the dynamic channel assignment, channel bonding, Multi-AP communication, Multiple input multiple output (MIMO), multi-user (MU) MIMO coordination, and enhanced handoff management.

The basic steps are illustrated below:

- indoor WLAN deployment
- simple replication algorithm for the distribution of mini-batch data on APs
- collaborative training among APs to update the GNN-edge based on local data, i.e., CGNN
- compressed collaborative training among APs to update GNN-edge based on local data, i.e., cCGNN

A. WLAN Indoor Deployment

The evaluations consider a WLAN deployment of 16 symmetric APs, i.e., all APs have equal opportunity to compete for the channel access. The CGNN-edge and cCGNN-edge

deployment seek to maximize the throughput with the help of joint carrier sensitivity threshold and transmit power control.

B. Evaluation Dataset

The procedure to collect the datasets is based on distributed learning of agents as discussed in Section IV-B. The 10 simulation experiments with 10,000 learning iterations are considered to collect the relevant data. In each iteration, a monitoring phase is used by an agent to monitor coexisting APs and report it back to a connected station. The number of coexistent APs vary between 2 to 16 in each iteration deployed in a $10 \times 5 \times 10m$ scenario.

C. Training

1) Training of GNN on the Cloud: The offline dataset is divided into two separate datasets. The first part is used as the initial part of the training dataset, while the second part is used to test the CGNN-edge during the training process. The testing part is used to validate the CGNN-edge and it determines the end of the CGNN-edge training and the beginning of the optimization process.

2) Training of CGNN and cCGNN: The agent can make a transition from the monitoring phase to the train phase to train the CGNN or cCGNN once the collection is complete. An agent can switch to a collection state at the next scheduled collection time. To distribute the initial training data across different APs for the warm-up training, we use a simple algorithm which takes the number of APs and assigns each AP a batch size of 50 from the test sets.

The training procedure on the edge considers online dataset as part of the training data set. After every optimization round, the collected dataset entry is appended to the local dataset of an AP. Accordingly, the CGNN and cCGNN are trained with an incremental training dataset in an online manner by using the proposed collaborative learning procedures as given in Algorithm 3 and Algorithm 4. This assures an adaptive behavior of the cCGNN and CGNN under dynamic WLAN deployments.

The branchynet AlexNet [18] is trained over the collected dataset. It has 5 early exit points, each with 12, 16, 19,20, and 22 layers. The per-layer latency is determined based on the input size, output size, features set, and model size.

D. Results

Figure 6 shows the change in accuracy for both the GNN-cloud and CGNN-edge as the number of APs are changed from 2 to 8 in steps of 2. Although, an addition of APs slows down the convergence rate, however, CGNN-edge can demonstrate similar accuracy for all network sizes. GNN-cloud can achieve better accuracy due to the availability of large data sets, however, CGNN-edge tries to achieve comparable accuracy. It can be established from this observation that the overall accuracy change will become negligible as more APs are added in the network.

Figure 7 evaluates the prediction accuracy of the trained CGNN-edge under different network sizes. It can be noted that varying the number of epochs has resulted in an accuracy improvement. The addition of APs has noticeable accuracy improvement with an increase in the number of epochs contrary to the first scenario.

Figure 8 shows the accuracy for CGNN-edge under different batch sizes. The CGNN performs better for large network size under growing batch size. A small batch size has a lower impact on a small network. However, this difference is prominent between small and large networks as the batch size grows. The larger the batch size an agent has, the less frequent gradient updates are required, thereby increasing the accuracy as evident.

The comparison of the AP configuration time with the CGNN-edge and GNN-cloud with an increase in the number of APs is illustrated in Figure 9. It is observed that the GNN-cloud has poor scalability. In practice, an increase in the number APs causes network delay between the APs and cloud resulting in a longer wait for configuration time. The CGNN-edge improves scalability better, compared with the GNN-cloud. This is mainly attributed to the accessible multiple edge nodes, low network delay, and higher inference speed.

Figure 10 evaluates the effect of batch size on the accuracy CGNN and cCGNN with a network size of 5 APs. It can be noted that an increase in batch size does not contribute to the accuracy of both the models. However, CGNN tends to demonstrate better accuracy, compared to the cCGNN. This is attributed to its frequent and accurate collaborative gradient updates, compared to the cCGNN with delayed and compressed updates.

Figure 11 shows average throughout for all the three schemes for different APs. The figure shows that all three schemes provide fair throughput. However, CGNN-edge demonstrates much more stable throughput compared to GNN-cloud. Due to local early and collaborative inference, both the CGNN-edge and cCGNN-edge results in less variance in throughput.

Figure 12 shows the communication overhead incurred by all three schemes as a function of network latency. This overhead includes the communication time taken by each scheme for the configuration. It includes either the communication time between APs for collaborative training both for the cGNN-edge and CGNN-edge or communication time between AP and cloud. It is observed that the cCGNN-edge has lowest communication time compared with the GNN-cloud and CGNN-edge. Local inference on the edge reduces the communication overhead significantly. However, this communication overhead becomes less significant with an increase in network latency as demonstrated in real-world scenarios with unstable network latency.

Loss convergence of cCGNN-edge is evaluated with drop ratios of 0 and 50. As illustrated in Figure 13, the cCGNN-edge approaches fast convergence where full gradient coordinates are exchanged, i.e., CGNN-edge. However, as shown in Figure 13, with only 50% gradient coordinates exchanges, convergence rate degrades significantly.

Figure 14 illustrates the satisfaction of APs for GNN-cloud, CGNN-edge, and cCGNN-edge with different latency requirements. The figure shows the dissatisfied latency re-

quirements with negative accuracy. All the three schemes are unable to satisfy the AP requests with latency requirements of $100ms$. However, both the CGNN-edge and cCGNN-edge satisfy all the requests exceeding latency requirements $200ms$ with reasonable accuracy. Due to the fast inference supported by the early exit and local inference, the proposed schemes can meet stringent latency requirements. However, GNN-cloud satisfies configuration requests with $500ms$ or higher latency requirements with better accuracy compared to CGNN-edge and cCGNN-edge.

Since the GNN-cloud uses all the GNN layers for inference, it results in better accuracy with higher latency, however, incurring higher communication overhead between edge hosts/APs and cloud. Therefore, GNN-cloud demonstrates better accuracy than CGNN-edge and cCGNN-edge under higher latency requirements.

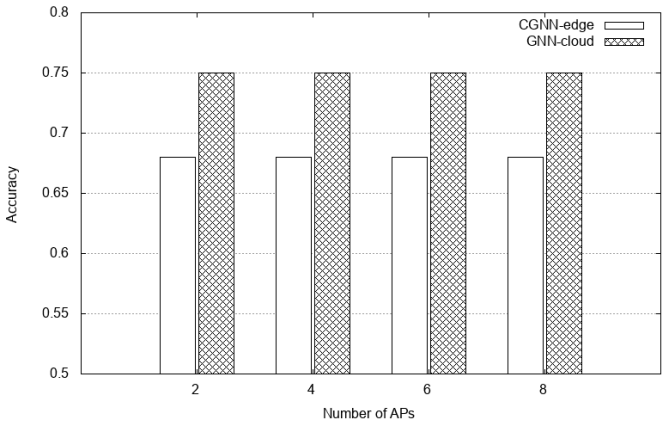

Fig. 6. CGNN-edge accuracy for APs.

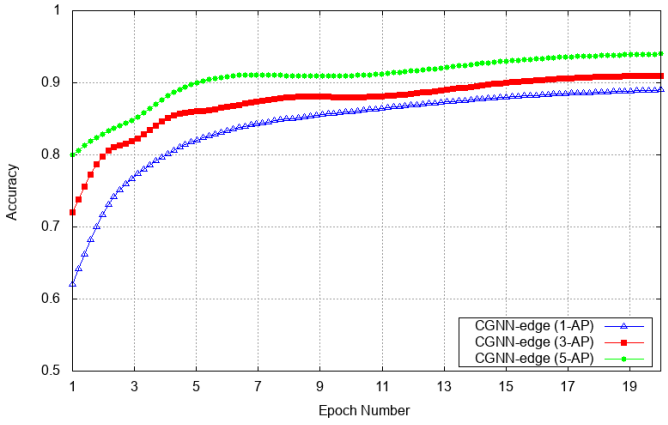

Fig. 7. CGNN-edge accuracy for epochs.

Figure 15 compares the learning time of the GNN-cloud with the CGNN-edge and cCGNN-edge. GNN-cloud runs on the CC. It utilizes centralized SGD for learning. As it is centralized it takes fewer epochs to achieve the desired accuracy. However, it requires frequent disk access to the training set which increases the overall learning time. CGNN-edge and cCGNN-edge running on 8 APs require a higher number of epochs to achieve the desired accuracy, however,

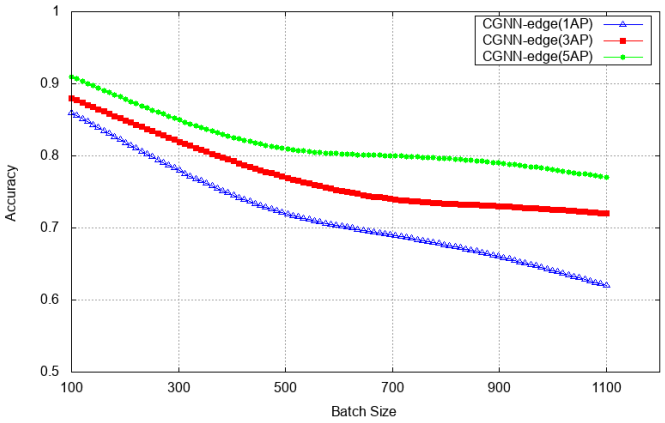

Fig. 8. CGNN-edge accuracy for batch size.

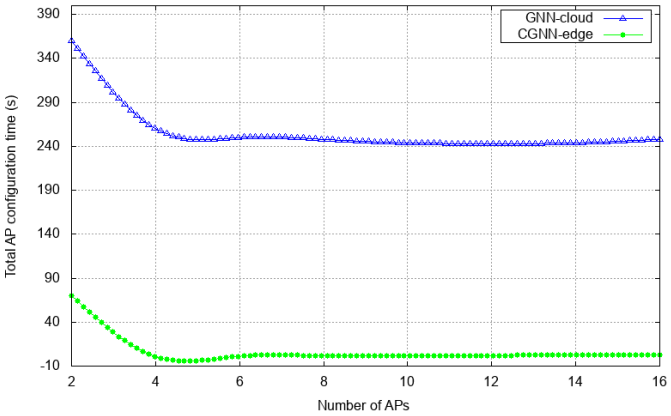

Fig. 9. CGNN-edge configuration time.

distributed learning reduces the learning time significantly as demonstrated in Figure 15. Properties including robustness and distributed learning make both the CGNN-edge and cCGNN-edge a suitable choice for edge inference.

Figure 16 compares the accuracy of CGNN-edge with and without collaborative learning. It can be observed from the Figure that the accuracy of CGNN-edge drops when APs stop exchanging gradient updates. In this case, APs rely on the local updates based on the SGD and do not take into consideration the configuration effects experienced by the neighboring APs.

Figure 17 compares the communication pattern of cCGNN-edge with CGNN-edge based on CC. Figure plots the average number of messages exchanged among nodes and the total number of messages received by the CC for 200 epochs. It can be observed that communication is balanced in cCGNN-edge based on CDSGD. On the other hand, CC receives a higher number of messages to update the SGD. It is apparent that the reliance on CC can limit the edge system scalability and performance.

With no doubt, CGNN-edge and cCGNN-edge must trade something over its offered advantages. Specifically, the edge nodes execute tasks to locally update training datasets, the computation load of APs is inevitably heavier on the account of local training process. This can result in higher energy cost

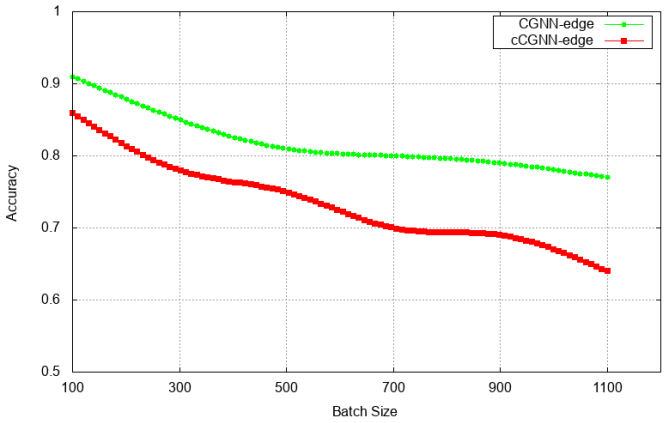

Fig. 10. CGNN-edge and cCGNN-edge accuracy for batch sizes.

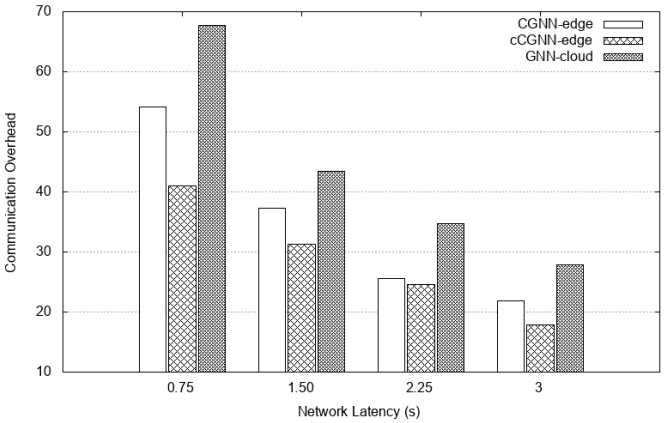

Fig. 12. CGNN-edge and cCGNN-edge communication overhead.

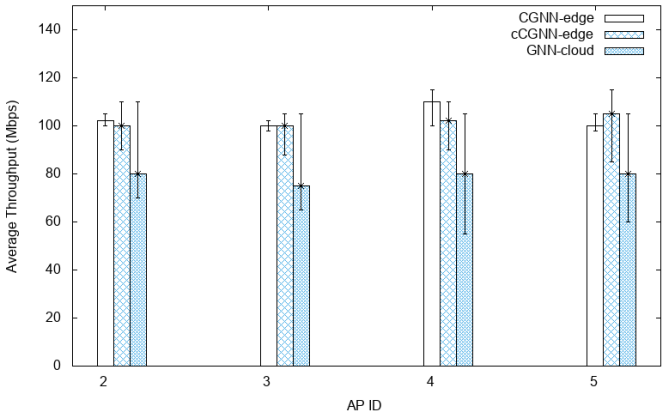

Fig. 11. Average throughput of cCGNN-edge, CGNN-edge, and GNN-cloud.

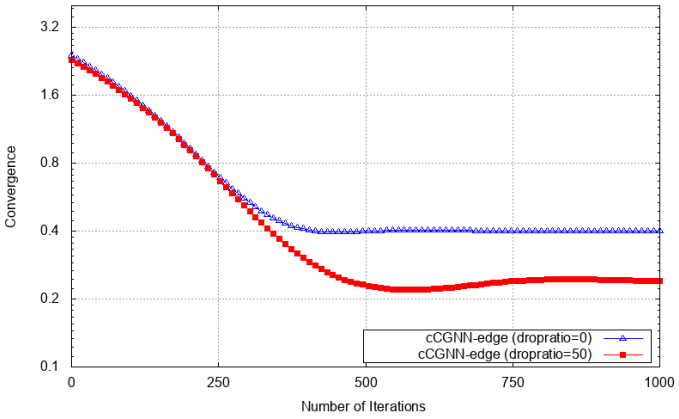

Fig. 13. cCGNN-edge convergence.

on edge devices, and these are open questions to address.

IX. Conclusion and Future Work

Topological dependencies and latency sensitivity are ongoing critical challenges faced by the denser mobile edge deployments. In this research, the proposed distributed collaborative deep learning approaches called CGNN-edge and cCGNN-edge optimize spectrum utilization and throughput with low latency inference thus meeting the requirements of time-critical applications. In both the approaches, lighter GNN model coupled with an early branching mechanism significantly accelerates the execution/prediction. The proposed collaborative learning mechanism based on SGD improves the accuracy and convergence of the CGNN-edge by exchanging stochastic gradients efficiently among edge devices, i.e., agents/APs. Further, the compressed collaborative learning mechanism adopted from the momentum correction reduces the frequency of gradient exchanges among agents, thereby reducing the communication overhead while still achieving comparable accuracy to the collaborative learning mechanism. The experimental results reveal that both the CGNN-edge and cCGNN-edge provides higher throughput, shorter response time, and better spectrum utilization under indoor dense WLAN deployments in comparison to cloud-based GNN prediction.

Our future work envisions to yield more interpretable model to extract edge dependencies based on self-attention [48]. Specifically, the future work will target drawing out global dependencies between input, output, and topology.

Acknowledgement

This work is supported by the Science Foundation of Ireland under Technology Innovation Development Award (TIDA) (grant number P2038 SFI 17/TIDA/5130).

References

[1] Jie Tang, Shaoshan Liu, Bo Yu, Weisong Shi,PI-Edge, ”A Low-Power Edge Computing System for Real-Time Autonomous Driving Services,” preprint, arXiv:1901.04978v1.

[2] S. Kim, D.-Y. Kim, J. -H. Park, ”Traffic management in the mobile edge cloud to improve the quality of experience of mobile video,” Comput. Commun. 2018, 118, pp. 40–49.

[3] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, “Content Popularity Prediction Towards Location-Aware Mobile Edge Caching,” arXiv preprint arXiv:1809.00232, Sep. 2018.

[4] S. Sarkar, S. Chatterjee, and S. Misra, “Assessment of the Suitability of Fog Computing in the Context of Internet of Things,” IEEE Trans. Cloud Comput., 2018, 6(1), pp. 46-59.

[5] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, “Cooperative edge caching in user-centric clustered mobile networks,” IEEE Trans. Mobile Comput., 2018, 17(8), pp. 1791-1805.

[6] V. Nae, A. Iosup, and R. Prodan, “Dynamic resource provisioning in massively multiplayer online games,” IEEE Trans. Parallel Distrib. Syst., 2011, 22(3), pp. 380-395.

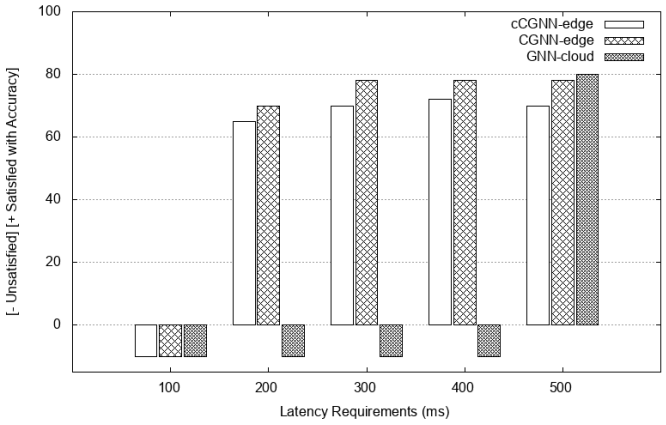

Fig. 14. Latency requirement satisfaction of cCGNN-edge, CGNN-edge, and GNN-cloud.

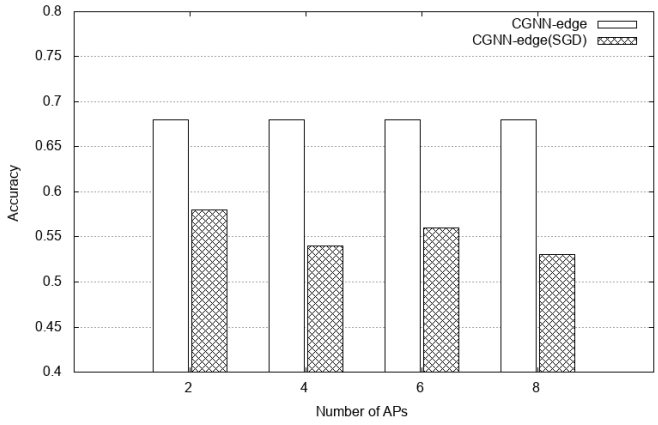

Fig. 16. Accuracy of CGNN-edge and CGNN-edge with SGD.

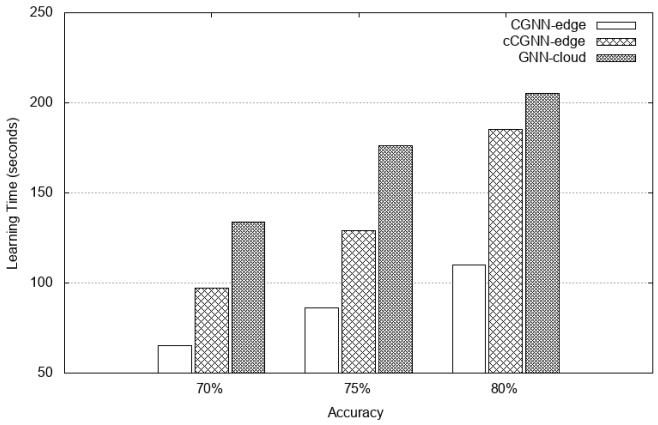

Fig. 15. Learning time of cCGNN-edge, CGNN-edge, and GNN-cloud.

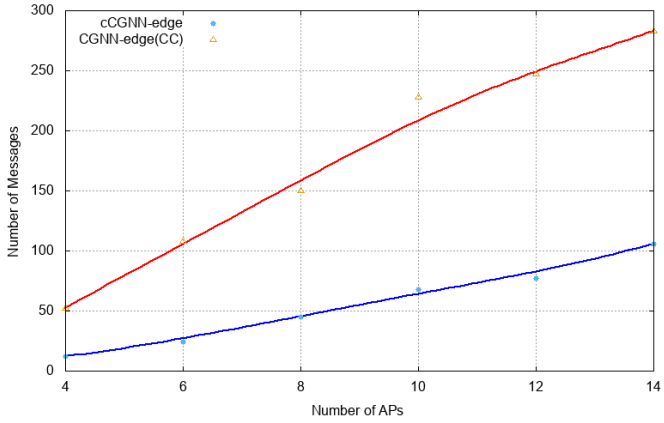

Fig. 17. Average number of messages/AP and total messages/CC.

[7] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," IEEE Commun. Surv. Tutor. 2018, 20, pp. 2923–2960.
[8] M.S. Mahdavinejad, M. Rezvan, M. Barekatain,P. Adibi, P. Barnaghi, A.P. Sheth, "Machine learning for internet of things data analysis: A survey," Digit. Commun. Netw. 2018, 4, pp. 161–175.
[9] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in the 37th IEEE International Conference on Distributed Computing Systems, 2017.
[10] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," IEEE/ACM Transactions on Networking, 2017, 25(5), pp. 2643–2654.
[11] E. Park, J. Ahn, S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in CVPR '17, USA, 2017.
[12] S. Han, J. Pool, J. Tran, W.J. Dally, "Learning both weights and connections for efficient neural networks," in NIPS'15,Canada,2015, pp.7–12.
[13] M. Verhelst, B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices," IEEE Solid-State Circuits Mag. 2017, 9, pp. 55–65.
[14] N.D. Lane, S. Bhattacharya, P. Georgiev,C. Forlivesi, L. Jiao, L. Qendro, F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in IPSN'16, 2016, Austria, pp. 11-14.
[15] H. Li, K. Ota, M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," IEEE Netw. 2018, 32, pp. 96–101.
[16] J. Konecn y, H. B. McMahan, F. X. Yu, P. Richt arik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," arXiv preprint arXiv:1610.05492, 2016.
[17] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE Trans. Neural Netw., 2009, 20(1), pp. 61–80.
[18] S. Teerapittayanon, B. McDanel, H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks", International Conference on Pattern Recognition, 2016.
[19] H. Khelifi, S. Luo, B. Nour, A. Sellami, H. Moungla,S. H. Ahmed, and M. Guizani, "Bringing deep learning at the edge of information-centric internet of things," IEEE Commun. Lett., 2018, pp. 1–1.
[20] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," in Advances in neural information processing systems, 2015, pp. 1756–1764.
[21] Y. Zhang, M. J. Wainwright, and J. C. Duchi, "Communication-efficient algorithms for statistical optimization," in Advances in Neural Information Processing Systems, 2012, pp. 1502–1510.
[22] C. Ma, J. Konecn y, M. Jaggi, V. Smith, M. I. Jordan, P. Richt arik, and M. Takac, "Distributed optimization with arbitrary local solvers," Optimization Methods and Software, 2017, 32(4), pp. 813–848.
[23] H. Li, K. Ota, K., M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," IEEE Netw. 2018, 32, pp. 96–101.
[24] X. Chen X, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," 2018, arXiv:1804.00514.
[25] M. Min, D. Xu, L. Xiao, Y. Tang, D. Wu, "Learning-based computation offloading for IoT devices with energy harvesting," 2017, arXiv:1712.08768.
[26] S. Han, J. Pool, J. Tran, W.J. Dally, "Learning both weights and connections for efficient neural networks," In Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS), 2015, Canada, pp. 7–12.
[27] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. arXiv, 2016, arXiv:1602.07360v3.

[28] J. Tang, D. Sun, S. Liu, J.L. Gaudiot, "Enable deep learning on IoT devices," Computer 2017, 50, pp. 92–96.
[29] Y. Wang, H. Li, X. Li, "Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices," In Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 2016, pp. 7-10.
[30] L. Du, Y. Du, Y. Li, J. Su, Y.-C Kuan, C.-C. Liu, M.-C.F.A. Chang, "Reconfigurable streaming deep convolutional neural network accelerator for internet of things," IEEE Trans. Circuits Syst. I Regul. Pap. 2018, 65, pp. 198–208.
[31] X. Chen, H. Zhang et al., "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," arXiv:1805.06146, Nov. 2018.
[32] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.
[33] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," arXiv preprint arXiv:1707.01083, 2017.
[34] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network,"in Design, Automation Test in Europe Conference Exhibition (DATE), 2017.
[35] Li, Dawei, et al. "DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition," Edge Computing (SEC), IEEE/ACM Symposium on. IEEE, 2016.
[36] S. Teerapittayanon, B. McDanel, H.T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," In IEEE International Conference on Distributed Computing Systems (ICDCS),2017, pp. 328–339.
[37] S. Bhattacharya and N. D. Lane, "Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables," In the 14th ACM Conf. Embedded Network Sensor Systems CD-ROM, ser. SenSys '16, 2016, pp. 176–89.
[38] M. A. Alsheikh et al., "Mobile Big Data Analytics Using Deep Learning and Apache Spark," IEEE Network, 2016, 30(3),pp. 22–29.
[39] C. Liu et al., "A New Deep Learning-Based Food Recognition System for Dietary Assessment on an Edge Computing Service Infrastructure," IEEE Trans. Services Computing. DOI: 10.1109/TSC.2017.2662008.
[40] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," Future Generation Computer Systems, 2018, 88,pp. 16–27.
[41] M. F. A. Jong Hwan Ko, Taesik Na and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," https://arxiv.org/pdf/1802.03835.pdf, 2018, arXiv Preprint.
[42] F. Wilhelmi, S. Barrachina-Munoz, B. Bellalta, C. Cano, A. Jonsson, and ˜G. Neu, "Potential and pitfalls of multi-armed bandits for decentralized spatial reuse in WLANs," Journal of Network and Computer Applications, 2019, 127, pp. 26–42.
[43] S ergio Barrachina-Munoz, Francesc Wilhelmi, Ioannis Selinis, and ˜ Boris Bellalta. Komondor: a Wireless Network Simulator for Next Generation High-Density WLANs. arXiv preprint arXiv:1811.12397,2018.
[44] Yurii Nesterov, "Introductory lectures on convex optimization: A basic course", Springer Science & Business Media, 87, 2013.
[45] Boris T Polyak, "Some methods of speeding up the convergence of iteration methods," USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.
[46] Xinghao Pan Jianmin Chen Rajat Monga Samy Bengio and Rafal Jozefowicz. "Revisiting distributed synchronous SGD", in: arXiv preprint:1702.05800, 2017.
[47] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training", In: arXiv:1712.01887, 2018.
[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," In Advances in Neural Information Processing Systems, 2017,pp. 6000–6010.
[49] B. Cao, L. Zhang, Y. Li, D. Feng and W. Cao, "Intelligent Offloading in Multi-Access Edge Computing: A State-of-the-Art Review and Framework," IEEE Communications Magazine, 2019, 57(3), pp. 56-62.
[50] Bin Cao, Shichao Xia, Jiawei Han, and Yun Li, "A Distributed Game Methodology for Crowdsensing in Uncertain Wireless Scenario", IEEE Transactions on Mobile Computing, 2019, 19(1), pp. 15-28.

Shagufta Henna is a research fellow with the CONNECT - the Science Foundation Ireland Research Centre for Future Networks and Communications based in the telecommunication software and systems group, Waterford institute of technology, Waterford, Ireland from 2018 to 2019. She received her doctoral degree in Computer Science from the University of Leicester, UK in 2013. She was an assistant professor from 2013 to 2018 at Bahria University, Islamabad, Pakistan. She has published several scientific papers in leading refereed journals and conferences. She is a senior member of the IEEE. She is currently serving the editorial boards of IEEE Access, EURASIP Journal on Wireless Communications and Networking, IEEE Future Directions, and Springer Human-centric Computing and Information Sciences. Her current research interests include deep learning, edge intelligence, network security, machine learning for 5G and beyond, and big data analytics.

Alan Davy received the B.Sc. (with Hons.) degree in applied computing and the Ph.D. degree from the Waterford Institute of Technology, Waterford, Ireland, in 2002 and 2008, respectively. Since 2002, he has been with the Telecommunications Software and Systems Group, originally as a student and then, since 2008, as a Post-Doctoral Researcher. He was a recipient of the Marie Curie International Mobility Fellowship in 2010, which brought him to work at the Universitat Politecnica de Catalunya for two years. He is currently Head of Department of Computer Science, Waterford institute of technology, Waterford, Ireland. His current research interests include Virtualised Telecom Networks, Fog and Cloud Computing, Molecular Communications and TeraHertz Communication.