IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 10 March 2023, accepted 26 March 2023, date of publication 29 March 2023, date of current version 6 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3262945

RESEARCH ARTICLE

Experimental Evaluation and Analysis of Federated Learning in Edge Computing Environments

PHAM KHANH QUAN, (Graduate Student Member, IEEE), MAJID KUNDROO, (Graduate Student Member, IEEE), AND TAEHONG KIM, (Senior Member, IEEE)

School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, South Korea

Corresponding author: Taehong Kim (taehongkim@cbnu.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2022R1I1A3072355.

ABSTRACT Federated learning (FL) is a machine learning system that allows a network of devices to train a model without centralized data. This characteristic makes FL an ideal choice for machine learning using user data while maintaining privacy. Many applications, such as autonomous vehicles that adapt to pedestrian behavior and smart healthcare devices, require machine learning to be performed in edge computing environments. However, the performance of FL in edge computing environments has received limited research attention, and previous studies have not comprehensively evaluated heterogeneity aspects, such as system heterogeneity, statistical heterogeneity, and communication bandwidth. This study aims to fill this research gap by conducting experimental evaluations and detailed analyses of FL in edge computing environments. Specifically, we set up an experimental testbed based on the FL framework on the KubeEdge platform and evaluate the diverse effects of heterogeneity aspects on the model convergence time such as homogeneous versus heterogeneous hardware resources, balanced versus imbalanced data distributions, duplicate versus nonduplicate datasets, client participation ratio at each round, the client selection algorithm, and the communication bandwidth. Overall, this study contributes to the advancement of the field of FL by expanding the understanding of its performance in edge computing environments and providing guidelines for efficient and effective FL systems in heterogeneous environments, which can ultimately benefit various industries and domains.

INDEX TERMS Federated learning, heterogeneity, Kubernetes, KubeEedge, edge computing, client selection.

I. INTRODUCTION

Federated learning (FL) is a collaborative and decentralized approach that takes advantage of large amounts of data from users for training while ensuring user privacy [1]. The core idea behind FL is decentralized learning [2] in which the user data is never sent to the central server. Conversely, each client trains the model locally using its own local data and updates local model parameters that are sent to the server. The server then obtains all the locally trained model parameters and effectively averages them to create an improved global model parameter. FL encounters several core challenges [3], such as system heterogeneity, statistical heterogeneity, and communication bandwidth.

System heterogeneity refers to the capacities of each device, which may vary depending on the local storage (memory), local computational resources (CPU), network connectivity (LTE, wifi, etc), and power (battery level). In practice, some devices might run out of battery or drop the wifi connection during training. Therefore, methods that address system

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

heterogeneity issues must satisfy the following requirements: 1) the server should automatically compute and select a subset of clients that have the highest hardware resources for training, and 2) the server should automatically disconnect from resource-constrained clients and low-bandwidth devices. For example, Nishio et al. [4] proposed a FedCS algorithm that can request that clients provide their resource information, such as wireless channel states, computational capacities, and training sample size, for determining which clients can participate in the training process.

Statistical heterogeneity indicates that the data distribution across clients is inherently non-independent and identically distributed (non-IID) [5]. Diverse methods have been proposed to tackle non-IID data in FL, which include data sharing [5], data augmentation [6], local fine-tuning [7], personalization layer [8], multi-task learning [9], knowledge distillation [10], lifelong learning [11], adaptive optimizers [12], and client clustering [13].

Communication bandwidth is a common issue in FL owing to diverse network connectivity such as 4G, 5G, and Wi-Fi, among devices. It highly affects model convergence in edge computing. In particular, model convergence can be significantly delayed if clients use low-bandwidth links. Therefore, the current research directions of FL must fulfill one of these two requirements to minimize costs: 1) reduce the size of messages transmitted per round; or 2) reduce the number of rounds required to achieve the desired accuracy,i.e. For instance, Nguyen et al. [14] proposed a fast-convergent FL algorithm to reduce the number of rounds to reach an expected accuracy or loss value by sampling devices intelligently.

FL is one of the most important applications for edge computing because many applications, including autonomous vehicles, chatbots, electronic payments, digital assistants, and recommendation tools, require artificial intelligence (AI) technology for analysis, detection, prediction, and categorization. These applications require machine/deep learning techniques to operate in edge computing environments [15]. Thus, edge computing is an important environment for FL; therefore, FL and edge computing are complementary to each other. Although several studies [16], [17], [18], [19], [20], [21], [22] have addressed FL in edge computing, we evaluate the performance of FL in edge computing environments. In particular, we investigate the diverse aspects of FL performance, including system heterogeneity, statistical heterogeneity, and communication bandwidth, and provide important lessons and insights for deploying the FL paradigm in edge computing environments.

In summary, this study has the following contributions:

- We proposed a KubeEdge-based FL framework that enables containerized FL algorithms to be deployed on a container orchestration platform. This framework is capable of deploying and managing FL algorithms to edge nodes that are geographically distributed with heterogeneous resources and data.
- We established experimental testbeds to evaluate the diverse effects of FL in an edge computing environment. To the best of our knowledge, this is the first study to comprehensively evaluate and analyze the performance of FL in an edge computing environment.
- We derived diverse lessons from the experimental performance analysis, which can be beneficial for optimizing edge-based FL systems. We also address research challenges, that can improve the model convergence time in edge computing environments of FL systems.

The remainder of this paper is organized as follows. Section II reviews related works on FL in edge computing environments. Section III briefly describes FL, Kubernetes, and KubeEdge platforms. Section IV describes our FL architecture in the KubeEdge platform and the key evaluation points. Section V analyzes the performance evaluation results and summarizes the lessons considering system heterogeneity, statistical heterogeneity, system statistical heterogeneity, and communication bandwidth. Finally, we conclude our study and present the scope for future works in Section VI.

II. RELATED WORKS

This section reviews related studies that evaluate FL in edge computing environments. For instance, survey papers [16], [17], [18] provide a comprehensive review and overview of the challenges and potential solutions for implementing FL in edge computing environments; however, they did not conduct a performance analysis to evaluate the effect of FL in edge computing environments.

Duan et al. [19] proposed a profile-based modeling framework and performance analysis method for evaluating the delay performance of edge-based FL systems. The main purpose of this study was to optimize the latency between client and server connections of FL in edge computing environments, but it has several limitations. This study assumed a homogeneous computational resources environment, which may not be representative of real-world scenarios where edge devices have diverse hardware and software configurations. In addition, the proposed model did not account for the impact of statistical heterogeneity, which can significantly affect FL performance.

Similarly, Bochie et al. [20] evaluated the impact of mobile network parameters on FL performance by simulating ideal conditions and actual wireless network conditions using the Flower framework. However, their experiments were constrained to a simulated environment in which the computational resources of each individual client could not be controlled. Furthermore, the study only evaluated the impact of system heterogeneity using the different failure probabilities applied to all clients to simulate that these clients were disconnected during the training, which may not fully capture the complexity and variability of mobile networks.

Likewise, Tahir et al. [21] surveyed the challenges of training FL models over heterogeneous data samples on IoT devices and analyzed the impact of system heterogeneity on IoT networks. The study also evaluated the effectiveness

of existing solutions for mitigating the negative impact of statistical heterogeneity in FL and conducted a comparative analysis of the top FL algorithms over a heterogeneous dynamic IoT network. This study concluded that the recently proposed solutions for mitigating this problem are not effective in practice. Nevertheless, this study did not evaluate the effect of network latency and bandwidth communication overhead, which can seriously affect the convergence speed of the model in FL.

Nilsson et al. [22] conducted an evaluation of three FL algorithms, FedAvg, CO-OP, and federated stochastic variance reduced gradient (FSVRG) on the MNIST dataset. This study aimed to compare the effectiveness of FL, a distributed machine learning method, that learns a global model by aggregating models trained locally on data-generating clients. They compared FedAvg with centralized learning and concluded that when IID data were used, both methods were practically equivalent, but the centralized approach outperformed FedAvg with non-IID data. However, their experiments encountered limitations as they used a balanced distribution in which all clients received the same amount of data, which does not represent the statistical heterogeneity found in real-world scenarios. Additionally, there was no mention of any FL or edge computing framework in their study.

To the best of our knowledge, this study is the first work to comprehensively address the limitations of the existing studies mentioned above. To set up the experimental edge computing testbed, we surveyed edge computing frameworks and designed the KubeEdge-based FL framework. For instance, we can deploy the FL model on clients inside a cluster infrastructure that calls a pod, enabling us to easily manage the computational resources allocated for FL training on each client. This ensures that the lessons derived from the experiments on the effects of system heterogeneity accurately reflect actual resource allocation conditions in the real world. We carefully investigated the diverse effect of system and statistical heterogeneity such as homogeneous versus heterogeneous hardware resources, balanced versus imbalanced data distributions, client participation ratio at each round, and client selection algorithm. We also analyzed the effects of network latency and bandwidth communication overhead, which can significantly affect the convergence speed of the model in FL.

III. PRELIMINARIES

A. FEDERATED LEARNING

FL is a technique for training machine/deep learning models on multiple devices without accessing private local data, thereby providing a novel approach to developing AI applications while ensuring security for user data. There are essentially four stages in the FL process:

- **Task initialization:** The server chooses a subset of available clients and sends a machine learning model to these clients such as convolutional neural networks (CNN), long short-term memory (LSTM), etc.
- **Local model training:** After receiving the model from the server, clients start training with locally stored data for several epochs.
- **Local model updating:** After these clients have finished the training process, each client updates its weighted value to the server.
- **Global model aggregation:** As soon as the server receives the weighted values from clients, it operates a chain of tasks aggregation, analysis, and evaluation to provide a new global model with better weight values.

All steps are continuously repeated until the model has achieved the expected accuracy level. One of the famous aggregation techniques is FedAvg [23], which is a weighted average of the individual clients' losses, derived as follows:

$$\min_{w} f(w) = \sum_{k=1}^{C\times K} \frac{n_k}{n} f_k(w^k), \tag{1}$$

where $f(w)$ is the overall global loss, C is a random sample fraction of selected clients, K is the total amount of clients, n_k is the local data size of client k, and $n = \Sigma_{k=1}^{C\times K} n_k$ is the total number of training samples from the selected clients.

The local loss value function $f_k(w^k)$ for each individual client is calculated as follows:

$$f_k(w^k) = \frac{1}{n_k} \sum_{i=1}^{n_k} f_i(w^k) = \frac{1}{n_k} \sum_{i=1}^{n_k} l(x_i, y_i, w^k), \tag{2}$$

where $f_k(w^k)$ is the local loss value function for client k, $f_i(w^k)$ is the loss value function for sample i, x_i is the data feature for sample i, y_i is the data label on sample i, and w^k is the weight value for client k. We typically utilize $f_i(w^k) = l(x_i, y_i, w^k)$, wherein the loss on the sample (x_i, y_i) is made with the weight value w on client k.

B. EDGE COMPUTING PLATFORMS

This subsection compares various edge computing platforms, including Docker Swarm [24], Apache Mesos [25], Kubernetes [26], and KubeEdge [27], and discusses the reasons for selecting KubeEge to evaluate the FL performance in edge computing environments.

Docker Swarm, designed by Docker, is the most compatible and natural way to orchestrate docker container infrastructures. However, Docker Swarm does not support auto-scaling or load-balancing; scaling must be done manually and load-balancing must be performed through third-party solutions, such as AWS Elastic Load Balancing [28].

Apache Mesos supports large-scale clustering and can easily scale up to ten thousand nodes without drawbacks [29]. It is an operating system that uses API for resource management and scheduling across multiple servers and cloud environments.

Kubernetes [26], also known as K8S, is an open-source container orchestration framework originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF) [30]. It was designed to manage large numbers of containers in various environments,

including physical servers, virtual machines, and cloud platforms. Kubernetes offers a range of useful features, including automatic scaling, load balancing, storage management, and resource management. It also provides a web-based user interface for managing and troubleshooting clusters. In addition, Kubernetes requires less third-party software than other container orchestration tools such as Apache Mesos or Docker Swarm. A Kubernetes cluster comprises at least one master node and several worker nodes. Pods are the smallest execution unit with containerized applications and are managed by the controller at the master node. However, Kubernetes is too heavy for edge scenarios [31] and must be installed on high-resource devices.

KubeEdge [27] is an open-edge computing platform founded on the Kubernetes extension, and it helps developers deploy and manage containerized applications in the cloud and edge using the same suitable platform. Owing to its lightweight size [32] (66MB footprint and 30MB running memory), it can be fully deployed on low-resource devices and low-bandwidth environments, and KubeEdge has emerged as a potential alternative platform for edge environments. Moreover, KubeEdge handles networking, deployment, and data synchronization between edge and cloud infrastructure; in addition, it is an outstanding platform for edge computing applications by providing these remarkable features:

- **Customization for the edge environment:** KubeEdge is a tool that helps developers build containerized applications using hypertext transfer protocol (HTTP) or message queuing telemetry transport (MQTT) protocols, particularly for use in edge computing scenarios. It does this by using the MQTT protocol for communication and synchronization between the cloud and edge nodes. MQTT is a lightweight messaging protocol that enables resource-constrained devices to exchange messages efficiently, even in low-bandwidth networks. By utilizing MQTT, KubeEdge simplifies the process of building and deploying container applications at the edge.
- **Scalability:** According to recent testing [33], KubeEdge has been shown to be capable of scaling to support up to 100,000 edge nodes, which is ten times more than the maximum number of nodes that can be managed by Apache Mesos [29]. Moreover, KubeEdge can manage a maximum of 1,000,000 active pods. In contrast, the Kubernetes documentation [34] states that it is not capable of scaling beyond 5,000 nodes or 150,000 pods.
- **Security:** KubeEdge also offers security mechanisms such as SPIFFE and SPIRE [35], to ensure that only verified edge nodes can participate in data transmission, which is essential for maintaining data security in FL. A recent security audit [36] found that, while several moderate vulnerabilities were identified, no critical security flaws were present in the exposed endpoints of KubeEdge or their connections to sensitive components of the architecture. These vulnerabilities were promptly addressed and resolved.
- **Machine learning and AI applications:** KubeEdge is well-suited for deploying machine learning algorithms, making it a good choice for use in combination with FL algorithms. Moreover, applying machine learning to edge computing is currently widely used by developers. Using machine learning at the edge not only reduces bandwidth costs and latency but also achieves better privacy and security guarantees because the program execution is run entirely inside of the container and the training data does not leave the device or edge location to be analyzed.

IV. KubeEdge-BASED FEDERATED LEARNING FRAMEWORK AND KEY EVALUATION POINTS

In this section, we introduce our FL architecture in the KubeEdge platform and propose the key evaluation points used in this study. Figure 1 describes an architecture that is a combination of the FL paradigm and the KubeEdge platform.

The cloud node is responsible for managing all connected edge nodes and is composed of three components: a Kubernetes module, KubeEdge's CloudCore, and FL-Cloud pod for the FL application. The edge nodes consist of KubeEdge's EdgeCore and an FL-Edge pod that runs FL applications.

The Kubernetes module has three important components: Kube-API server, Kube-scheduler, and Kube-service. Kube-API server is a cluster gateway that brings the initial requests of any updates into the cluster; therefore, it acts as a gatekeeper for authentication to ensure that only authorized requests pass through to the cluster. Kube-scheduler is responsible for scheduling the application pods on different nodes based on the current workload and available resources on each node. Kube-service is a service that represents a persistent stable IP address used to access a specific pod; in addition, it provides a unique DNS name to handle a set of pods, which can achieve load balancing among these pods. Furthermore, Kube-service provides a static port on the node itself, which can be used to access the edge nodes externally and is known as NodePort. It is set to 30008 because NodePort can only be in a valid range, which by default is from 30000 to 32767.

In the KubeEdge module, CloudCore and EdgeCore are the two main submodules for data synchronization between cloud and edge nodes. CloudCore is a Kubernetes pod that operates inside the cloud node. The two key components of CloudCore are EdgeController and CloudHub. The EdgeController controls data from the cloud node to specific edge nodes and handles connections between the Kube-API server and EdgeCore, while the CloudHub works as a web socket connection to EdgeHub to update the changes from the cloud node to the edge nodes. EdgeCore is a group of components that establishes API-driven communication to CloudCore. EdgeCore consists of EdgeHub, Edged, ServiceBus, and MetaManager. EdgeHub is an equivalent implementation compared to CloudHub, and it handles the

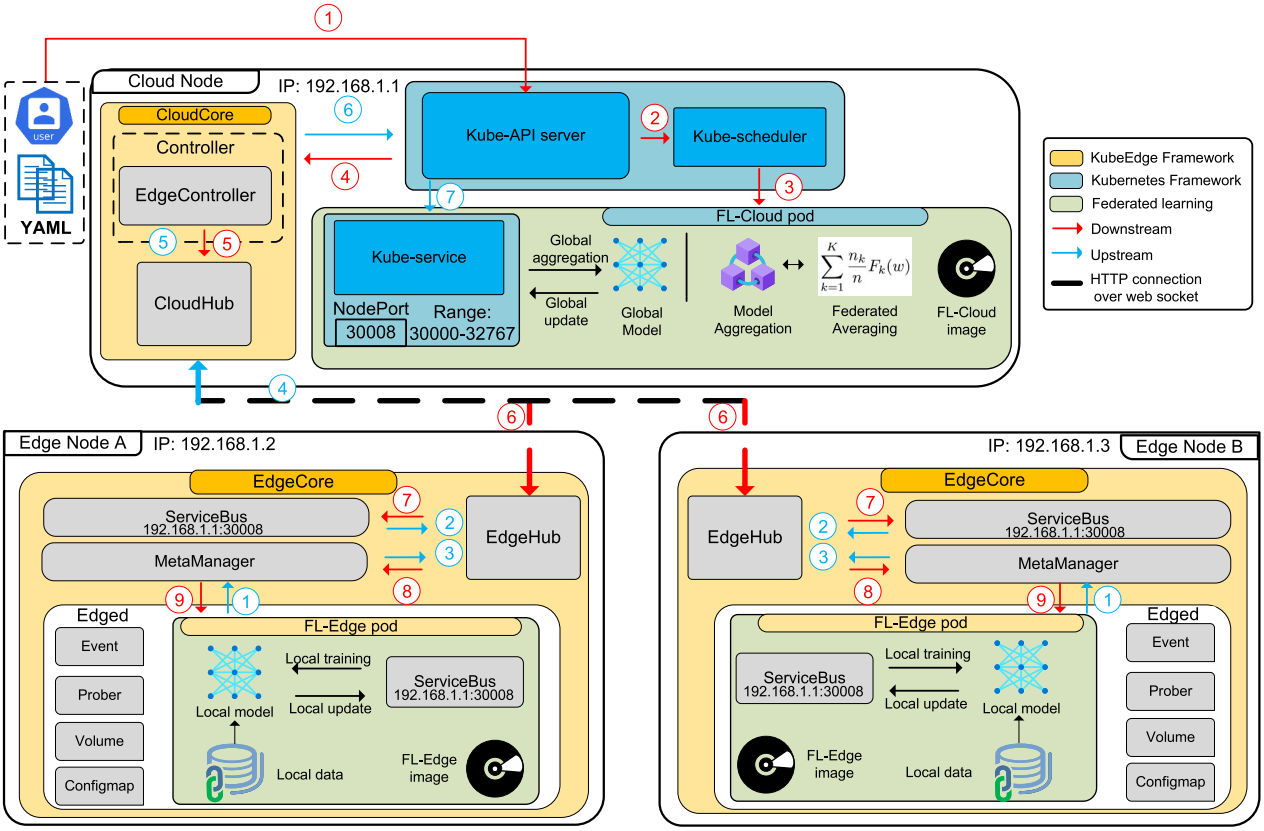

FIGURE 1. Federated learning (FL) in the KubeEdge architecture.

cloud-side resource updates and edge-side status changes as a web socket. Edged is a lightweight Kubelet implementation used to manage containerized applications running on edge devices. Edged comprises sub-module,s such as pod, event, probe, config map, and volume, to handle the containerized workloads or pod application life cycle. ServiceBus is an HTTP client that allows edge nodes to communicate with the HTTP server on the cloud node. MetaManager provides node-level metadata persistence functionalities, such as insert, update, delete, query, response, and node connection operations between Edged and EdgeHub.

There are two types of pods in FL applications: FL-Cloud and FL-Edge. FL-Cloud is an FL pod created to run on a cloud node. This pod contains the basic input parameters for the FL algorithm, which include round number, batch size, epoch, learning rate, sample fraction, a minimum number of participant clients, and the cloud node's IP address. FL-Edge is a federated learning pod created to run on edge nodes. In addition, the entire system is managed through a YAML file structure in which we can optionally initialize, delete, or monitor the training progress of FL applications.

Before deploying the FL-Cloud pod on the cloud node and the FL-Edge pods on the edge nodes, we need to create docker images for each of them. A docker image is a binary file that contains an application and all its dependencies, and the FL-Cloud and FL-Edge images include the entire source code of this FL application and necessary libraries written in Python. We used the Flower framework [37] to deploy the FL application, and therefore the sizes of the FL-Cloud and FL-Edge images will be equivalent to the size of the Flower framework. However, other open-source FL frameworks, such as TensorFlow Federated (TFF) [38], Federated AI Technology Enabler (FATE) [39], or LEAF [40], could also be used depending on the needs of the developers. In the KubeEdge framework, the operation of FL consists of two main phases: downstream and upstream.

In the downstream phase, the initialization process starts when we run the command **kubectl apply ***.yaml**. Then, the Kube-API server checks this command to see if it is correct. Next, the Kube-API server sends this message to Kube-Scheduler to create an FL-Cloud pod with all the information provided in the YAML file and containing a built-in container FL-Cloud image. Meanwhile, the Kube-API server also sends information to EdgeController to create the FL-Edge pod on each specific edge node with all the information listed in the YAML file such as the input parameters, cloud node's IP address, edge node name, local data distribution, and resource allocation for each pod via CloudHub-EdgeHub connections. At edge nodes, as soon as EdgeHub receives these messages from CloudHub, EdgeHub

TABLE 1. Resource distributions for system heterogeneity, statistical heterogeneity, system statistical heterogeneity, and communication bandwidth environments.

System Heterogeneity		Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7	Edge 8	Edge 9	Edge 10
Homogeneous	CPU	4	4	4	4	4	4	4	4	4	4
	Balance 1 (Non-Duplicate)	3000	3000	3000	3000	3000	3000	3000	3000	3000	3000
Heterogeneous	CPU	16	8	4	2	1	16	8	4	2	1
	Balance 1 (Non-Duplicate)	3000	3000	3000	3000	3000	3000	3000	3000	3000	3000
Statistical Heterogeneity		Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7	Edge 8	Edge 9	Edge 10
CPU		4	4	4	4	4	4	4	4	4	4
Balance 1 (Non-Duplicate)		3000	3000	3000	3000	3000	3000	3000	3000	3000	3000
Balance 2 (Duplicate)		4000	4000	4000	4000	4000	4000	4000	4000	4000	4000
Imbalance 1 (Non-Duplicate)		12000	2000	2000	2000	2000	2000	2000	2000	2000	2000
Imbalance 2 (Non-Duplicate)		4000	4000	3500	3500	3000	3000	2500	2500	2000	2000
Imbalance 3 (Duplicate)		18000	16000	14000	12000	10000	8000	6000	4000	2000	1000
Imbalance 4 (Duplicate)		10000	9000	8000	7000	6000	5000	4000	3000	2000	1000
System and Statistical Heterogeneity		Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7	Edge 8	Edge 9	Edge 10
CPU		16	8	4	2	1	16	8	4	2	1
Imbalance 3 (Duplicate)		18000	16000	14000	12000	10000	8000	6000	4000	2000	1000
Imbalance 4 (Duplicate)		10000	9000	8000	7000	6000	5000	4000	3000	2000	1000
Communication Bandwidth		Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7	Edge 8	Edge 9	Edge 10
CPU		4	4	4	4	4	4	4	4	4	4
Balance 1 (Non-Duplicate)		3000	3000	3000	3000	3000	3000	3000	3000	3000	3000

has the responsibility to transmit these messages to the other modules in EdgeCore such as ServiceBus and MetaManager. After that, EdgeCore stores the cloud node's IP address and the NodePort number in the ServiceBus. Meanwhile, EdgeHub sends messages to MetaManager informing it that a new FL-Edge pod needs to be initialized with the parameters listed in the YAML file. Next, based on the information provided by MetaManager, the Edged module creates the FL-Edge pod with the built-in container FL-Edge image. Finally, the FL-Edge pod is deployed inside the Edged module using the task initialization and local model training steps that are explained in Section III-A.

In the upstream phase, the Edged module reports the FL-Edge pod status, pod condition, and pod resource information to MetaManager. After the local training process, the local weight values in the FL-Edge pod are sent directly to the ServiceBus that contains the cloud node's IP address and NodePort number. Next, EdgeHub sends this information to EdgeController via the EdgeHub-CloudHub connections. During the monitoring and updating process, the EdgeController informs the Kube-API server about changes in resource status (node, pod, and configuration map changes). In the following step, the Kube-API server aggregates and analyzes the overall status of all FL-Edge pods on the edge nodes. This allows developers to monitor them easily when running **kubectl get pod**. Aside from forwarding the weight data from the edge nodes, the Kube-API server also sends this data to the FL-Cloud pod via KubeService. Finally, the FL-Cloud pod proceeds with the step mentioned in the global model aggregation section in Section III-A.

This study mainly focuses on evaluating the impact of heterogeneity in the edge computing environment, which affects the performance of FL via system heterogeneity, statistical heterogeneity, system statistical heterogeneity, and communication bandwidth, as summarized in Table 1.

To evaluate the influence of system heterogeneity on the model convergence in FL, we compared the homogeneous setting using identical CPU resources and the heterogeneous setting using different CPU resources, while distributing the same amount of data to each client. Then, we dropped the clients with low CPU resources from the training process to determine if this approach improves the model convergence.

Statistical heterogeneity in the model convergence was evaluated by comparing the effects of balance and redundancy in the data distribution. In addition, we also compared the effects of different client selection methods, such as random selection and top-data selection to determine a suitable selection method.

System statistical heterogeneity is a combination of system and statistical heterogeneity that more closely reflects real-world environments with significant differences in computational power and training samples across clients. We defined the influence level based on CPU resources and local training samples to quantify the resources of each client in terms of system statistical heterogeneity. Then, by selecting a certain fraction of clients based on their level of influence, we evaluated the model convergence time according to the client selection policy.

To evaluate the effect of communication bandwidth on the convergence of the model in FL, we distributed many types of bandwidths to 10 edge nodes while utilizing the same amount of CPU resources and constant data distribution across clients.

V. PERFORMANCE EVALUATION RESULTS

The evaluations are divided into four categories: system heterogeneity, statistical heterogeneity, system statistical heterogeneity, and communication bandwidth. The environmental settings of each category are summarized in Table 1. The FashionMNIST dataset [41], which consists of a training set of 60,000 samples and a testing set of 10,000 samples, is used

in these experiments, and the detailed experimental settings will be discussed at the beginning of each subsection.

A. SYSTEM HETEROGENEITY

To evaluate system heterogeneity, we divided the experiment into two categories: homogeneous and heterogeneous. While homogeneous settings allow all clients to have the same data and CPU resources, heterogeneous settings allow all clients to have the same amount of data but different CPU resources. It should be noted that we set the data distribution among clients non-duplicated to each other.

1) HOMOGENEOUS SYSTEM

We measured the model convergence time and the number of rounds required until a 91% accuracy was achieved in a homogeneous environment. In addition, we also evaluated the effect of the sample fraction of clients on the model convergence by randomly choosing 100, 80, 60, 40, and 20% of clients among 10 total clients for training in each round. The 100% fraction of clients indicates that all 10 edge nodes must join the training process, whereas the 80, 60, 40, and 20% sample fractions of clients indicate that only eight, six, four, and two edge nodes out of the 10 edge nodes are randomly selected for training in each round, respectively.

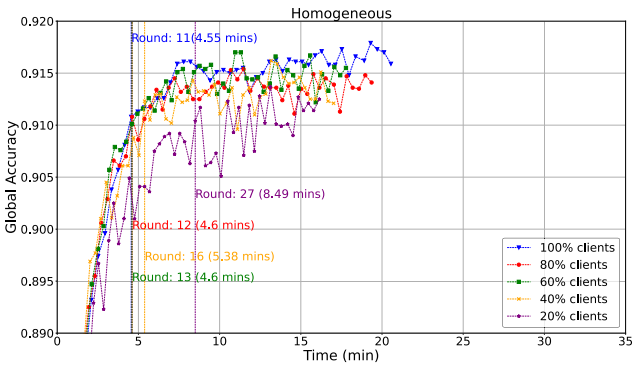

FIGURE 2. Global accuracy over time under homogeneous settings.

Figure 2 shows that under homogeneous settings, both the number of rounds and time required to reach 91% accuracy are decreased when the sample fraction of clients is increased. For example, the use of 100% of all clients requires only 11 rounds to achieve 91% accuracy, while the use of 80, 60, 40, and 20% of all clients needs 12, 13, 16, and 27 rounds, respectively, to achieve the same accuracy. Meanwhile, only 4.55 min is required to achieve 91% accuracy for 100% of clients while the use of 80, 60, 40, and 20% of all clients require 4.6, 4.6, 5.38, and 8.49 min respectively. This is because the total amount of data used in one round is reduced when a low sample fraction of clients are used. At 100%, a total of 30,000 training samples are used in each round, meanwhile, with 80%, 60%, 40%, and 20%, only 24,000, 18,000, 12,000, and 6,000 training samples are used in each round, respectively. These findings indicate that there is an inverse relationship between the sample fraction of clients and model convergence under homogeneous settings.

2) HETEROGENEOUS SYSTEM

Similar to the homogeneous settings, we evaluated the model convergence in a heterogeneous environment. In addition, the sample fractions of clients were selected according to the CPU resources. For example, the 100% sample fraction of clients indicates that all 10 edge nodes must join the training process, while for the 80% sample fraction of clients, edges 5 and 10 with 1 CPU resource are not chosen to join the training process. Similarly, for the 20% sample fraction of clients, only edges 1 and 6 with 16 CPU resources are chosen to join the training process.

However, with the 20% sample fraction of clients, only 6,000 identical samples are used during training from edges 1 and 6, which makes it impossible for the model to achieve 91% accuracy. Therefore, we shuffle the training samples after each round, which means that edges 1 and 6 have 3,000 different samples out of a total of 30,000 samples in each round. Because the edge devices are trained on data that is collected/generated after the previous round, this setting makes our system function closer to real-world systems.

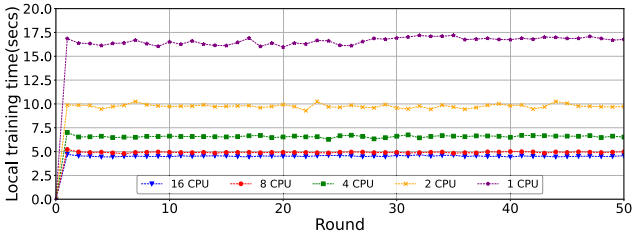

FIGURE 3. Local training time for each round for different CPU resources.

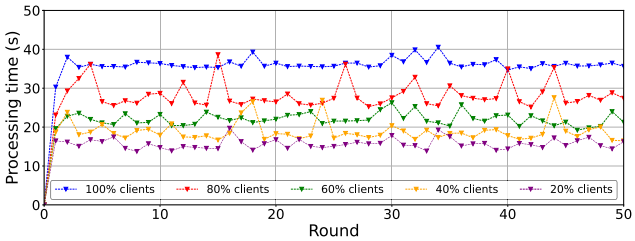

FIGURE 4. Processing time for each round under heterogeneous settings.

Figure 3 shows the effect of low CPU resources on the local training time for each round. For example, clients with 16 CPU resources require <5 s to complete the local training tasks, while clients with 1 CPU resource require >17 s to complete similar tasks.

Figure 4 shows the effect of low CPU resources on the processing time for each round. With the 100% fraction of clients, the lowest CPU clients (1 CPU) are included, and with the 80% fraction of clients, the second-lowest CPU clients (2 CPU) are included. From Figure 5, the 100% fraction of

TABLE 2. Convergence time (min), convergence round, and processing time (s) of random and top data selection under statistical heterogeneity.

Fraction of client	Imbalance 3 (Random)			Imbalance 3 (Top data)			Imbalance 4 (Random)			Imbalance 4 (Top data)		
	Round	Time	Process	Round	Time	Process	Round	Time	Process	Round	Time	Process
100%	4	4.302	64.539	4	4.302	64.539	6	3.819	37.0237	6	3.819	37.0237
80%	4	3.703	55.5515	3	2.878	58.5629	7	4.332	35.2743	6	3.602	35.3547
60%	4	3.155	47.3297	3	2.838	56.7741	7	3.869	33.1632	5	3.17	34.4496
40%	**4**	**3.019**	**45.2984**	**3**	**2.677**	**53.5457**	**7**	**3.54**	**30.3454**	**5**	**2.88**	**33.7614**
20%	7	4.06	34.8724	3	2.344	46.8987	14	6.375	27.322	29	18.515	31.7218

clients requires nearly 40 s to complete one round, while the 20% fraction of clients with the highest CPU resources requires approximately 17 s to complete the same task. Therefore, we can observe that the processing time on each round depends on the local training time of the lowest-CPU client.

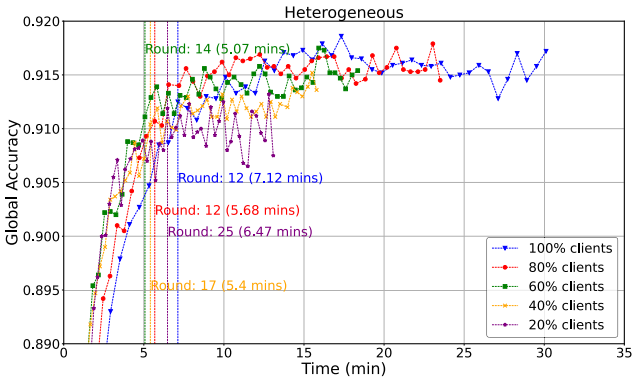

FIGURE 5. Global accuracy over time under heterogeneous settings.

Figure 5 shows that the 60% fraction of clients with heterogeneous settings provides the best convergence time with 5.07 min at round 14. Moreover, even though the 100% and 80% fractions of clients achieved 91% accuracy at round 12 due to their high amount of training samples in each round, they required more time to complete their local training process (7.12 min and 5.68 min, respectively). This is because the 2 and 1 CPU clients are used in the training progress for this system. In contrast, the 40% and 20% fractions of clients utilize only clients with high CPU resources (16 and 8 CPU) during the training process such that the training time for one round is relatively short. However, due to the relatively small amount of training samples in each round, more rounds are required to achieve 91% accuracy. Therefore, 5.4 min at round 17 and 6.47 min at round 25 are required when using the 40% and 20% fractions of clients, respectively. In particular, neither the 100% nor 20% fractions of clients achieve the best convergence time. In practice, the CPU distribution among clients may greatly vary, and a suitable client selection ratio for reducing the model convergence time remains an open issue.

B. STATISTICAL HETEROGENEITY

We evaluated the effects of statistical heterogeneity including balanced/imbalanced data and duplicate/non-duplicate data. For example, Balance 1 contains a non-duplicate data distribution among clients, whereas Balance 2 has a duplicate data distribution among clients. Similarly, Imbalance 1 and Imbalance 2 have non-duplicate data distributions among clients, while Imbalance 3 and Imbalance 4 have duplicate data distributions among clients. In addition, we also analyzed the effect of client selection on statistical heterogeneity by comparing two methods: random selection and top data selection.

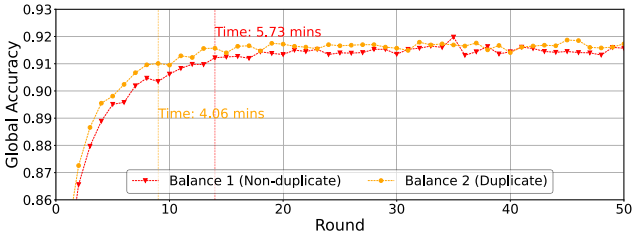

FIGURE 6. Convergence time and convergence round in the Balance datasets.

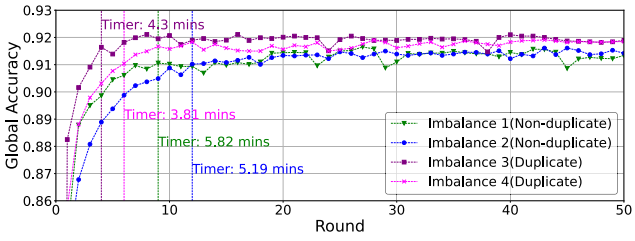

FIGURE 7. Convergence time and convergence round in the Imbalance datasets.

1) DUPLICATE DATA

Figures 6 and 7 show that the distribution of duplicate data over clients improved the model convergence regardless of balanced and imbalanced data under the data heterogeneity settings. For instance, Balance 1 (non-duplicate) requires 5.73 min to reach 91% accuracy compared to Balance 2 (duplicate), which required 4.06 min. Similarly, Imbalance 1 and Imbalance 2 (non-duplicate) required 5.82 and 5.19 min, respectively, to reach 91% accuracy compared to 4.3 and 3.81 min for Imbalance 3 and Imbalance 4 (duplicate). Therefore, the convergence time and convergence round required to achieve suitable accuracy can be improved by sharing public or similar data with clients in the real-world application of FL.

TABLE 3. Convergence time (min), convergence round, and processing time (s) using the client's influence level for system and statistical heterogeneity.

	α = 0.1 (Top CPU)					α = 0.5					α = 0.9 (Top Data)				
Imbalance 3	100%	80%	60%	40%	20%	100%	80%	60%	40%	20%	100%	80%	60%	40%	20%
Time (min)	**6.423**	**3.926**	**3.276**	**2.44**	**5.611**	6.423	5.549	2.922	2.556	4.097	6.423	4.094	3.88	**2.892**	2.49
Round	3	4	4	4	**11**	3	3	3	3	8	3	3	3	3	**3**
Process (sec)	128.4	58.8	49.1	**36.6**	30	128.4	110.9	58.4	51.1	30.5	128.4	81.8	76.2	**57.8**	48.1
Imbalance 4	100%	80%	60%	40%	20%	100%	80%	60%	40%	20%	100%	80%	60%	40%	20%
Time (min)	**5.507**	**4.068**	**3.103**	**2.864**	**10.16**	5.507	4.738	4.041	3.27	9.87	5.507	4.738	4.519	**3.936**	6.291
Round	6	6	6	7	**25**	6	6	6	6	24	6	6	6	6	**14**
Process (sec)	55.07	40.6	31.03	**24.5**	24.4	55.07	47.3	40.4	32.7	24.6	55.07	47.3	45.1	**39.3**	26.9

2) CLIENT SELECTION

To evaluate the effect of client selection in statistical heterogeneity, we compared random and top data client selection with the Imbalance 3 and Imbalance 4 distributions. The random selection process randomly chooses 100, 80, 60, 40, and 20% of clients among 10 clients for training in each round, while the top data selection method chooses clients according to the amount of training sample that each client possesses. For example, for the 100% sample fraction of clients, all 10 edge nodes must join the training process, while in the 80% sample fraction of clients, edges 9 and 10 with the lowest training samples are not chosen to join the training process.

In Table 2, we can see that the top data selection method can reduce the convergence time and convergence round compared to the random selection method. For example, in Imbalance 3, the 40% client fraction from random selection needs 4 rounds and 3.019 min to reach 91% accuracy while the top data selection method clients only need 3 rounds and 2.677 min. In Imbalance 4, random selection needs 7 rounds and 3.54 min to reach 91% accuracy while top data selection only needs 5 rounds and 2.88 min. This is because the clients with a larger amount of data serve to speed up the training process in each round. However, the processing time in each round for the top data selection method is higher than that for the random selection method. For example, a 40% fraction of random selection clients in Imbalance 3 only needs 45.29 s to complete one round while the same fraction of top data selection requires 53.54 s. Corresponding to Imbalance 4, the random selection clients only need 30.34 s to complete one round of training while the top data selection clients need 33.76 s. This is because the top data selection method trains with more training samples than the random selection method. This experiment demonstrates that in a statistical heterogeneity setting, using the top data method for client selection can lead to faster model convergence compared to random selection, because the clients with more training data serve to increase training speed in each round. However, it is worth noting that the top data selection method requires more processing time than the random selection method due to the larger amount of training data that is involved.

C. SYSTEM AND STATISTICAL HETEROGENEITY

To evaluate real use cases of the investigated system, we set all clients to have different CPU resources and different amounts of data and then ranked the clients based on their influence level, which is calculated as follows:

$$InfluenceLevel = \alpha(Data/1000) + (1-\alpha)CPU \quad (3)$$

where $\alpha \in [0, 1]$ is an adjustment factor, *Data* is the total amount of training sample of that client, and *CPU* is the number of cores possessed by each client. $\alpha \to 0$ means that any client with high CPU resources has a large influence level, while $\alpha \to 1$ indicates that clients with a large amount of data have a large influence level. Therefore, in an environment where both devices and data are heterogeneous, we can adjust α according to the client selection policy. In this experiment, we use the influence level of each client with three α values: 0.1(referred to as Top CPU), 0.5, and 0.9(referred to as Top Data).

In Table 3, the convergence time decreases when the fraction of clients decreases regardless of the client selection policy with all three α values. For example, in Imbalance 3 with $\alpha = 0.1$, a 100% fraction of clients needs 6.42 min to reach 91% accuracy while the 80, 60, and 40% fractions need 3.92, 3.27, and 2.44 min, respectively. In Imbalance 4 with $\alpha = 0.1$, the 100% fraction of clients requires 5.5 min to reach 91% accuracy while the 80, 60, and 40% client fractions require 4.06, 3.1, and 2.86 mins, respectively. However, at a certain client fraction, the convergence time significantly increases because the training data from the selected clients are not adequate to reach 91% accuracy. For example, the convergence time of the 20% client fraction was increased to 5.611 min in Imbalance 3 and 10.16 min in Imbalance 4. Therefore, we can conclude that determining an optimal setting for the client fraction is an important issue in the application of the investigated system.

Table 3 shows that the top CPU-based policy with $\alpha = 0.1$ exhibits a better convergence time than the top data-based policy with $\alpha = 0.9$. For instance, the 40% client fraction of the top CPU-based policy requires a minimum convergence time of 2.44 and 2.86 min for Imbalances 3 and 4 respectively. Therefore, it is recommended that the top CPU-based policy be used for most of the client fractions to achieve adequate convergence speeds. However, we should be noted that the use of a small client fraction in the top CPU-based policy may exhibit much worse convergence due to a lack of training data, as shown by the 20% client fraction example. Otherwise, the top data-based policy does not show a sudden increase in the convergence time even when the

TABLE 4. Latency, throughput, processing time and convergence time among 1 Gbps, 100 Mbps, 45 Mbps, 10 Mbps, 4 Mbps, and 1.544Mbps bandwidths.

Bandwidth	**1Gbps**	100Mbps	45 Mbps	**10Mbps**	4Mbps	**1.544Mbps**
Throughput (MB/s)	48.1557	2.5603	0.6355	0.1239	0.0482	0.0186
Average report time in each round (s)	**0.2757**	3.3747	10.2209	**51.6328**	132.64	**343.8**
Average processing time in each round (s)	**20.0035**	23.6739	29.8094	**71.865**	152.538	**365.4582**
Average processing time excluding the reporting time (s)	19.7278	20.2992	19.5885	20.2323	19.898	21.6532
Ratio of the reporting time in each round (%)	**1.378**	14.254	34.287	**71.846**	86.955	**94.073**
Total convergence time (min)	**4.046**	4.803	6.725	**15.864**	36.254	**68.075**

client fraction is low, and this system can be used to obtain stable, but not minimal, results regardless of the client fraction that is used.

D. COMMUNICATION BANDWIDTH

This section analyzes the effect of the communication bandwidth by applying different bandwidths of 1 Gbps, 100 Mbps, 45 Mbps, 10 Mbps, 4 Mbps, and 1.544 Mbps while using the same training data and CPU resources for all clients.

In Table 4, the average reporting time in each round is measured by calculating the average time sent from edge nodes to the cloud node until the model reaches 91% accuracy. Because the reporting time depends on the bandwidth between the edge nodes and the cloud node, when this bandwidth is higher, the average reporting time in each round will take less time. For example, the average reporting time in each round requires 0.2757 s at a 1 Gbps bandwidth, 51.63 s at a 10 Mbps bandwidth, and 343.8 s at a 1.544 Mbps bandwidth.

The communication bandwidth also significantly affects the total convergence time, which was 4.04 min at a 1 Gbps bandwidth, 15.86 min at a 10 Mbps bandwidth, and 68.075 min at a 1.544 Mbps bandwidth. Simultaneously, the average processing time in each round is increased to 20 s at a 1 Gbps bandwidth, 71.865 s at a 10 Mbps bandwidth, and 365.4582 s at a 1.544 Mbps bandwidth.

The ratio of the reporting time in each round can be calculated as follows:

$$\phi = \frac{\mu_R}{\mu_P} * 100, \tag{4}$$

where ϕ is the ratio of the reporting time (%), μ_R is the average reporting time in each round (s), and μ_P is the average processing time in each round (s).

Table 4 shows that the ratio of the reporting time in each round increases at decreasing bandwidths. For example, 1.37% is achieved at the 1 Gbps bandwidth, 71.846% at the 10 Mbps bandwidth, and 94.073% at the 1.544 Mbps bandwidth. This indicates that the connection bandwidth between the client and server can significantly affect the convergence time. Therefore, communication bandwidth is an important aspect to consider when designing and operating FL in edge computing environments.

E. SUMMARY OF LESSONS

This section provides a summary of the key lessons learned regarding the various factors that impact the convergence speed and efficiency of FL. Specifically, we discuss the impacts of system heterogeneity, statistical heterogeneity, system statistical heterogeneity, and communication bandwidth on the performance of FL systems. This section provides insights on how to choose the appropriate client selection policy and communication bandwidth to optimize the convergence speed and stability of the FL system, depending on specific application requirements.

1) SYSTEM HETEROGENEITY

- Under homogeneous settings, increasing the sample fraction of clients decreases both the number of rounds and the time required to achieve the desired accuracy. Therefore, an inverse relationship exists between the sample fraction of clients and the convergence speed of the model in homogeneous settings.
- Under heterogeneous settings, the processing time of each round depends on the local training time of the slowest client in terms of CPU performance. However, the distribution of CPU performance among clients can vary significantly in practice, making it challenging to determine an optimal client selection ratio to minimize the model convergence time. Therefore, identifying suitable client selection strategies remains an issue in such scenarios.

2) STATISTICAL HETEROGENEITY

- Making duplicate data among clients can enhance the convergence time and the number of rounds required to achieve the desired accuracy in real-world applications of FL.
- In statistical heterogeneity settings, selecting clients with the highest amount of training data can lead to faster model convergence compared with random selection, as those clients can provide more training data and thus increase the speed of training in each round. However, it is important to note that the top data selection method requires more processing time than the random selection method because of the larger amount of data involved. Therefore, careful consideration should be given to the selection strategy used in such scenarios to achieve optimal performance.

3) SYSTEM AND STATISTICAL HETEROGENEITY

- In an environment where both devices and data are heterogeneous, we can choose the clients according to the CPU resources and training samples of each client.

Choosing clients based on their CPU resources is effective for most client fractions to achieve adequate convergence speed. However, it is important to note that the use of a small client fraction may result in significantly worse convergence owing to the lack of training data. On the other hand, choosing clients based on the number of training samples shows stable results even if the client fraction is low, without a sudden increase in convergence time although it can not provide the minimum convergence time. Therefore, the selection of an appropriate client selection policy requires careful consideration of the trade-offs between convergence speed and stability, depending on the specific application requirements.

4) COMMUNICATION BANDWIDTH

- When a higher connection bandwidth is utilized, the total convergence time, average reporting time, and average processing time in each round is reduced. As a result, communication bandwidth is a crucial factor to consider when designing and implementing FL systems in edge computing environments.

VI. CONCLUSION

In this study, we designed a KubeEdge-based FL framework and demonstrated its benefits. We developed experimental testbeds to evaluate the effects of FL in edge computing environments. This study also provided a comprehensive evaluation of FL performance in edge computing environments under various types of heterogeneity, including system heterogeneity, statistical heterogeneity, and communication bandwidth. Our results showed that these aspects have a significant impact on the model convergence of FL in edge-computing environments.

Our findings contribute to the advancement of the FL field by providing guidelines for optimizing model convergence in heterogeneous environments. Using these guidelines, developers can build more efficient and effective FL systems in edge computing environments, which can benefit various industries and domains. Our study also highlights the need for further research in this area to develop more robust and scalable FL systems that can handle the challenges of heterogeneity in edge computing environments. Overall, this study underscores the potential of FL in edge computing and the importance of addressing the challenges of heterogeneity for its successful deployment.

REFERENCES

[1] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017, *arXiv:1712.07557*.
[2] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "BrainTorrent: A peer-to-peer environment for decentralized federated learning," 2019, *arXiv:1905.06731*.
[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020, doi: 10.1109/MSP.2020.2975749.
[4] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.
[5] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*.
[6] M. A. Tanner and W. H. Wong, "The calculation of posterior distributions by data augmentation," *J. Amer. Statist. Assoc.*, vol. 82, no. 398, pp. 528–540, 1987.
[7] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–29.
[8] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global representations," 2020, *arXiv:2001.01523*.
[9] Y. Zhang and Q. Yang, "An overview of multi-task learning," *Nat. Sci. Rev.*, vol. 5, no. 1, pp. 30–43, Jan. 2018.
[10] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, pp. 1789–1819, Mar. 2020.
[11] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4555–4562, Oct. 2019.
[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Feb. 2011.
[13] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-IID data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–9.
[14] H. T. Nguyen, V. Sehwag, S. Hosseinalipour, C. G. Brinton, M. Chiang, and H. V. Poor, "Fast-convergent federated learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 201–218, Jan. 2021.
[15] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and T. Kim, "ElasticFog: Elastic resource provisioning in container-based fog computing," *IEEE Access*, vol. 8, pp. 183879–183890, 2020.
[16] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.
[17] H. G. Abreha, M. Hayajneh, and M. A. Serhani, "Federated learning in edge computing: A systematic survey," *Sensors*, vol. 22, no. 2, p. 450, 2022.
[18] A. Brecko, E. Kajati, J. Koziorek, and I. Zolotova, "Federated learning for edge computing: A survey," *Appl. Sci.*, vol. 12, no. 18, p. 9124, 2022.
[19] Q. Duan and M. Roshanaei, *Modeling and Performance Analysis on Federated Learning in Edge Computing*. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, 2021, pp. 41–46.
[20] K. Bochie, "An analysis of federated learning on mobile networks," *Netw. Comput. Appl.*, vol. 194, pp. 103213–103258, 2021.
[21] M. Tahir and M. I. Ali, "On the performance of federated learning algorithms for IoT," *IoT*, vol. 3, no. 2, pp. 273–284, Apr. 2022.
[22] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proc. 2nd Workshop Distrib. Infrastruct. Deep Learn.*, Dec. 2018, pp. 1–8.
[23] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2016, pp. 1273–1282.
[24] S.-I. Riadi and A. Sugandi, "Forensic analysis of Docker Swarm cluster using GRR Rapid Response framework," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 2, pp. 1–8, 2019.
[25] G. Rattihalli, P. Saha, M. Govindaraju, and D. Tiwari, "Two stage cluster for resource optimization with Apache Mesos," in *Proc. 10th Workshop Many-Task Comput. Clouds, Grids, Supercomputers (MTAGS)*, 2019, pp. 1–8.
[26] V. Medel, O. Rana, J. Ángel Bañares, and U. Arronategui, "Modelling performance & resource management in kubernetes," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, 2016, pp. 257–262.
[27] L. H. Phuc, M. Kundroo, D.-H. Park, S. Kim, and T. Kim, "Node-based horizontal pod autoscaler in KubeEdge-based edge computing infrastructure," *IEEE Access*, vol. 10, pp. 134417–134426, 2022.
[28] *Elastic Load Balancing*. Accessed: Jan. 25, 2023. [Online]. Available: https://aws.amazon.com/elasticloadbalancing
[29] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, and S. Shenker, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement.*, 2011, pp. 295–308.
[30] *Make Cloud Native Ubiquitous*. Accessed: Jan. 25, 2023. [Online]. Available: https://www.cncf.io/

[31] C. Mahler. *Kubernetes on the Edge: Getting Started With Kubeedge and Kubernetes for Edge Computing*. Accessed: Jan. 25, 2023. [Online]. Available: https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing
[32] S. K. D. Huawei and J. Du (Huawei). *Kubeedge, a Kubernetes Native Edge Computing Framework*. Accessed: Jan. 25, 2023. [Online]. Available: https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro/
[33] W. Xu. (2022). *Test report on kubeedge's support for 100,000 Edge Nodes*. [Online]. Available: https://kubeedge.io/en/blog/scalability-test-report/
[34] *Considerations for Large Clusters*. Accessed: Jan. 25, 2023. [Online]. Available: https://kubernetes.io/docs/setup/best-practices/cluster-large
[35] T. Geer. *Secure Kubeedge Using Spiffe/Spire*. Accessed: Jan. 25, 2023. [Online]. Available: https://kubeedge.io/en/blog/secure-kubeedge
[36] D. Korczynski and A. Korczynski. *Kubeedge Holistic Security Audit Engagement*. Accessed: Jan. 25, 2023. [Online]. Available: https://adalogics.com/blog/kubeedge-security-engagement
[37] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2020, *arXiv:2007.14390*.
[38] *Tensorflow Federated: Machine Learning on Decentralized Data*. Accessed: Jan. 25, 2023. [Online]. Available: https://www.tensorflow.org/federated
[39] *An Industrial Grade Federated Learning Framework*. Accessed: Jan. 25, 2023. [Online]. Available: https://fate.fedai.org/
[40] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konecný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, 2018, pp. 1–9.
[41] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

PHAM KHANH QUAN (Graduate Student Member, IEEE) received the B.S. degree in electronics and telecommunications engineering from Ton Duc Thang University, Ho Chi Minh City, Vietnam, in 2020. He is currently pursuing the M.S. degree in information and communication engineering with Chungbuk National University, South Korea. He was a Student Research Assistant with the Wireless Communications Research Group, Ton Duc Thang University, under the supervision of Dr. Phuong T. Tran. His major research interests include edge computing, edge AI, federated learning, the IoT applications, and container orchestration.

MAJID KUNDROO (Graduate Student Member, IEEE) received the M.S. degree in computer science from the Islamic University of Science and Technology, Jammu and Kashmir, India, in 2019. He is currently pursuing the Ph.D. degree with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include edge computing, edge AI, the Internet of Things, and federated learning.

TAEHONG KIM (Senior Member, IEEE) received the B.S. degree in computer science from Ajou University, South Korea, in 2005, and the M.S. degree in information and communication engineering and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 2007 and 2012, respectively. He was a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT) and the Samsung DMC Research and Development Center, from 2012 to 2014. He was also a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), South Korea, from 2014 to 2016. Since 2016, he has been an Associate Professor with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include edge computing, container orchestration, the Internet of Things, and federated learning. He has been an Associate Editor of IEEE Access, since 2020.