IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 December 2023, accepted 28 December 2023, date of publication 4 January 2024,
date of current version 11 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3349691

RESEARCH ARTICLE

FedOps: A Platform of Federated Learning Operations With Heterogeneity Management

JIHWAN MOON, SEMO YANG, AND KANGYOON LEE, (Member, IEEE)
Department of Computer Engineering, Gachon University, Seongnam-si 13120, South Korea

Corresponding author: KangYoon Lee (keylee@gachon.ac.kr)

This work was supported in part by the Commercialization's Promotion Agency for Research and Development Outcome (COMPA) Grant funded by the Korean Government (MSIT) under Grant 2022-Future Research Service Development Support-1-SB4-1, and in part by the National Research Foundation of Korea (NRF) funded by MSIT under Grant NRF-2022R1F1A1069069.

ABSTRACT Federated learning (FL) is a decentralized machine learning (ML) method that enables model training while preserving privacy. FL is gaining attention because it avoids data transfer to the server, facilitating the decentralized learning of the traditional ML model. Despite its potential, FL project is significantly more challenging to develop than centralized ML methods owing to decentralized local data. We propose *FedOps*, federated learning operations for constructing systematic FL project by enhancing machine learning operations (MLOps) to be effectively applied to FL while preserving its core process. To address complexity of FL implementation, we developed *FedOps platform*, which involves *FedOps*-based projects to manage the whole lifecycle in FL context. We also investigated methods to identify performance degradation factors in FL and suggest an approach for improvement. *FedOps Platform* provides an analysis tool for client heterogeneity, called *chunk-bench*. This tool enables researchers and engineers to gain insights into systems heterogeneity by using only small chunk of the clients' data to execute test in the shortest time possible while tracking the systems heterogeneity across the clients. By addressing systems heterogeneity, *FedOps Platform* achieved 13%–43% improvement in communication cost-to-accuracy and 20%–68% improvement in time-to-accuracy. We believe that *FedOps Platform* offers an optimal solution for end-to-end development of FL projects, with significantly improving both computational and communication efficiencies.

INDEX TERMS Cloud-native application, federated learning, FedOps, MLOps, privacy, non-IID data, systems heterogeneity.

I. INTRODUCTION

Federated learning (FL) is a decentralized machine learning (ML) method that enables model training while preserving privacy. Recent data-driven services collect private user information, including sensitive biometric data, leading to concerns regarding privacy protection. Therefore, the demand for FL, as a privacy-preserving method, rises significantly and it is not limited to certain application fields.

Despite the potential of FL, organizations without FL-specialized researcher struggle to implement FL service owing to its complexity and inferior performance compared to centralized ML. Specifically, FL performance varies significantly based on the device condition or local data distribution. The difficulty level of model updates, optimization, and verification is significantly higher than that of general ML methods, owing to its inability to share client data to the server [1]. To address these issues, a holistic guideline is necessary for developing FL projects.

Given that the data and model are not centralized in FL, the complexity of implementing FL project is exacerbated by inevitable challenges like heterogeneous clients, model updates aggregation, and personalization. To address these challenges, systematic operations like machine learning operations (MLOps) are necessary for the entire FL lifecycle management. MLOps is a methodology that defines how to

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

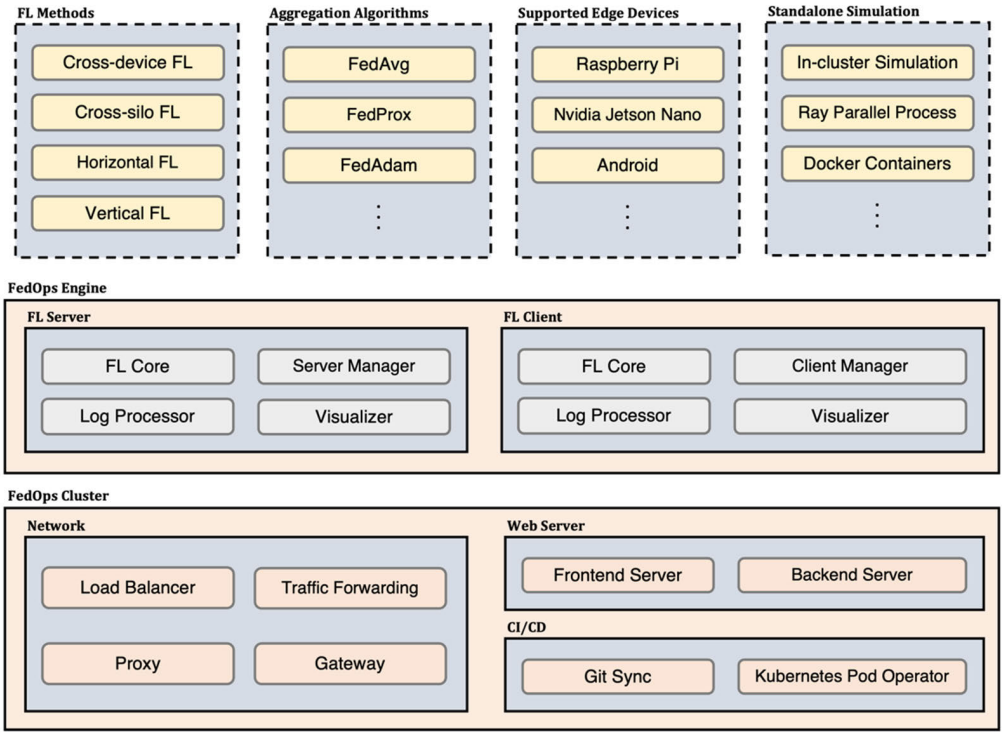

FIGURE 1. Overview on FedOps Platform implementation.

track and manage the full lifecycle of ML from data collection to model training, testing, and monitoring. However, unlike traditional ML workflow, there are several restrictions to direct adaptation of MLOps in FL lifecycle. There are some difficulties which have not seen in centralized ML, such as hyperparameter optimization, feature engineering, model selection, and debugging, thereby making direct application of MLOps difficult [2]. Our goal is to build an MLOps-like platform which is generalized to be applicable across a wide range of application fields. We expect our work to reduce the barriers to entry for FL projects for organizations and researchers.

We propose *FedOps*, federated learning operations for constructing systematic FL project by enhancing MLOps to be effectively applied to FL while preserving its core process. By designing the architecture of *FedOps*, we aim to address the following research questions:

(i) What components, functions, and interactions from MLOps are required for *FedOps* to support systematic implementation of FL?

(ii) How can we identify the factors that cause degraded performance during the FL process, and which techniques can be used to improve the degraded learning performance?

In this study, we used Kubernetes and several containerized cloud-native applications to develop a system architecture that collaboratively functions and makes the platform extensible. For researchers and engineers who may not be specialized in FL, we designed *FedOps Platform,* which is abstracted for ease of use to ensure that FL service can be seamlessly applied to on-premise server or cloud environments. *FedOps Platform* was developed to facilitate *FedOps*-based projects, enabling automation and monitoring of the entire FL process—from integration to test, deployment, and beyond—while providing an optimized development and production environment. Fig. 1 shows that *FedOps Platform* provides cloud-native *FedOps* modules with FL core engine and several FL methods. These include network load balancer, web interface, and FL model aggregation algorithms that support various simulation methods for real edge devices.

Additionally, we show a *chunk-bench* method to mitigate unpredictable factors (e.g., systems heterogeneity) of FL clients, thereby improving the performance of FL process. The *chunk-bench* method immediately analyzes the heterogeneity of the clients during the initial communication round of local model training. We demonstrated the experiment of setting an adaptive threshold in client selection by using the insights gained by *chunk-bench*. We expect *FedOps Platform* to support streamlined operational development and research strategies in FL. To check code of *FedOps Platform* and to reproduce experiments, please see https://github.com/kumass2020/FedOps-Chunk-Benchmark.

The main contributions of this study are as follows:

- We propose *FedOps* methodology; It includes system architecture designed to consider the entire process of the FL lifecycle. The module and pipeline composition that comprise the system and its roles are introduced.
- We present *FedOps Platform*, which is designed to easily deploy FL services using cloud-native applications and manage the entire FL lifecycle.

- As a core tool of the *FedOps Platform*, we propose *chunk-bench*—which provides a heterogeneity analysis strategy for analyzing hardware performance and communication cost. According to the insights gained from *chunk-bench*, we present an experiment that improves the time-to-accuracy and communication costs.

II. BACKGROUND AND RELATED WORKS

This section describes background information on FL and client heterogeneity, which is one of the major challenges in FL. Additionally, we introduce related studies on FL frameworks and MLOps, which motivated the development of *FedOps Platform*.

A. FEDERATED LEARNING

In centralized ML, all data converges to a single host, accumulating problems related to privacy, efficiency, and scalability [3]. FL offers a novel paradigm that resolves the disadvantages by training ML models using decentralized local model training and model aggregation.

Typically, FL aims to the generalization of global model and the personalization of local model [4]. Generalized global model has the capacity to handle unseen data from heterogeneous clients. Personalization enables local model to optimize for client-specific features.

To collaboratively merge numerous local model updates into the global model, an aggregation strategy is necessary for global model optimization. The most popular aggregation algorithm is FedAvg. This algorithm uses the average of local model updates from clients in the server [5]. FedAvg aims to minimize $\min_w f(w) = \sum_{k=1}^{N} p_k F_k(w)$. Here, w is the model parameter, and N is the client number. When the data is partitioned, p_k is the data ratio $\frac{n_k}{n}$ for client k for the total data number $n = \sum_k n_k$ [6]. We used FedAvg as a baseline method for our experiment to demonstrate how to mitigate FL performance degradation factors.

During the communication process of model updates, secure aggregation can be applied to maintain the integrity of the model. It is implemented by adding secure protocols or cryptographic techniques that apply encryption to model updates [7]. *FedOps Platform* provides a secure connection by adding Google remote procedure call (gRPC), which is a secure protocol based on transport layer security (TLS).

B. CLIENT HETEROGENEITY

Client heterogeneity denotes the disparities stemming from distinct client environments in FL. It is unpredictable and uncontrollable, thus posing a significant challenge in FL. Client heterogeneity is generally classified into data heterogeneity and systems heterogeneity [1].

Data heterogeneity is referred to as statistical heterogeneity, which means the imbalance in local data deriving from the non-independent identically distributed (non-IID) data that the clients have. Non-IID data could appear owing to the data characteristics such as feature distribution, label distribution, and data quantity skewness [1]. To improve the global model performance while mitigating data heterogeneity, attempts such as vertical FL [8] were made through changes in data concatenation. For the personalization (also known as local fine-tuning) of the local model, there have been attempts to apply model agnostic meta-learning (MAML) [9].

Systems heterogeneity refers to the varying capabilities for task processing due to the factors such as hardware performance, network connectivity, storage, and other client-specific conditions [6]. To mitigate systems heterogeneity, the client state should be periodically updated, and policies such as cut-off should be established to handle the cases such as failing to learn multiple times or having excessively slow model training speed. Additionally, policies that compress model size or reduce the number of communications (e.g., fewer communication rounds and numerous local epochs) can be adjusted to reduce communication cost [10]. Existing FL frameworks have difficulties at finding performance degradation factors and directly regularize them [11], [12], [13]. Addressing systems heterogeneity issue is one of the main purposes of this study. We will continue the relevant discussion in Sections III and IV.

C. FEDERATED LEARNING FRAMEWORKS

FedOps Platform has been influenced by active studies on FL frameworks. These studies improved accessibility to FL research but have some limitations on handling scalability, communication, cloud-native support and the whole lifecycle management. Table 1 shows an overview of comparison with key properties of those FL frameworks.

FedML [11], an open-source FL framework supports creating projects of cross-device FL and cross-silo FL methods through an easily accessible Web UI. However, FedML does not support cloud-native applications which make the FL project extensible and scalable. Also, FedML could not complete simulation over 100 clients in a single cluster.

FATE [12] was developed as an open-source framework for supporting the FL ecosystem, and it provides sub-services such as FATEFlow which supports the FL end-to-end pipeline and kubeFATE which uses Kubernetes and cloud-native applications. However, FATE does not provide Web UI and Python library that is independently executable.

Further, LEAF [13] provides statistical system metrics and datasets for FL benchmark environment. FedBalancer [14] selects FL clients based on metadata created from results of local model optimization to enhance the FL performance. And other frameworks like Flower [15], FLSim [16], TensorFlow federated (TFF) [17], FLUTE [18], and Clara [19] were implemented through interfacing the existing ML libraries such as PyTorch and TensorFlow. These frameworks abstract the general ML code level to provide a testbed for researchers, offering simulation environments for numerous client instances. Although these frameworks showed comparable works in specific tasks, they did not

TABLE 1. Comparison of federated learning frameworks.

	TFF	LEAF	FATE	FedScale	FedML	FedOps
Simulation	V	V	V	V	V	V
Scalability	-	-	-	O	-	V
Heterogeneous systems	O	-	V	V	V	V
Heterogeneous data	O	-	V	V	V	V
Communication overhead	-	-	O	O	-	V
Cloud-native platform	-	-	O	-	-	V
FL lifecycle management	-	-	-	-	-	V

O: planned or partially implemented

```python
from fedops.server import app
from fedops.server import server_utils
import init_gl_model

if __name__ == "__main__":
    # Read server config file
    config_file_path = './config.yaml'
    config = server_utils.read_config(config_file_path)

    # Dataset Name
    dataset = config['data']['name']

    # Build model
    model, model_name = build_gl_model(dataset)

    # Load Data
    x_val, y_val = load_data(dataset)

    # Start fl server
    fl_server = app.FLServer(config, model, model_name, x_val, y_val)
    fl_server.start()
```

FIGURE 2. An example code snippet of executing FedOps server.

consider constructing the whole FL lifecycle using their frameworks.

We propose *FedOps Platform* to construct and manage a whole FL lifecycle upon an easy-to-use Web UI and Python library. Fig. 2 shows how simply *FedOps* server can be executed using Python library interface. In *FedOps* simulation setting, example dataset is automatically partitioned and distributed to clients by data loader, which is less complex than most FL frameworks.

D. MLOPS

MLOps is a methodology for managing ML lifecycle from requirements to model tuning, culminating in data processing via a refined workflow or pipeline. Implementing MLOps requires considering the degree of automation for each model-serving and process pipeline [20]. Additionally, it is also referred to as the methodology for effectively developing the ML service for business requirements [21]. Generally, MLOps is implemented within a microservice architecture, where containerized applications serve as the individual components of the system, and automated pipelines are used to manage the workflow [22].

Since FL is a more complex form of centralized ML, the workflow management performed in MLOps would not function optimally in FL. There were some approaches to adapt MLOps to FL, but they are partially implemented and not scalable in the FL lifecycle. Therefore, we had to reorganize the following MLOps components to suit *FedOps*: (i) continuous integration (CI), continuous deployment (CD), continuous testing (CT), (ii) data extraction, analysis and (iii) model serving, testing, validating, deploying [23].

III. FEDOPS

We present the *FedOps* architecture that manages the entire FL lifecycle with best practices, as well as presenting some considerations for performing *FedOps*.

A. FL SYSTEM RESTRAINTS

In FL, factors that have not been considered in ML are presented such as the heterogeneous environments and communication. Therefore, the following factors must be additionally considered in *FedOps*: (i) communication efficiency, (ii) systems heterogeneity, (iii) data heterogeneity (non-IID data), (iv) privacy preservation, and (v) aggregation and optimization strategy [8], [24]. The complexity of these challenges that FL has faced has become the primary motivation for constructing *FedOps Platform*.

Fig. 3 shows the overall system architecture and compositional factors of *FedOps*. The architecture is composed of the server and client lifecycles and subcomponents that constitute each lifecycle. We reorganized existing MLOps components with added FL components for handling FL process. The key components for the FL process are highlighted in the Fig. 3.

B. CLIENT LIFECYCLE

Typically, MLOps does not consider the client-side model training; however, given that the model training of FL primarily takes place at the client-level, managing the model lifecycle of clients becomes essential. As indicated in Sections C-1 to C-4 of Fig. 3, client lifecycle in *FedOps* is divided into four sections.

The function and composition of submodules are as below:

1) DEPLOYMENT

Build and update of the client instance should be implemented with the code that is specific to the target device, considering the processor architecture which is referred to as the instruction set such as AMD64, ARM64, and ARMV7. For example, in mobile devices such as smartphones, using

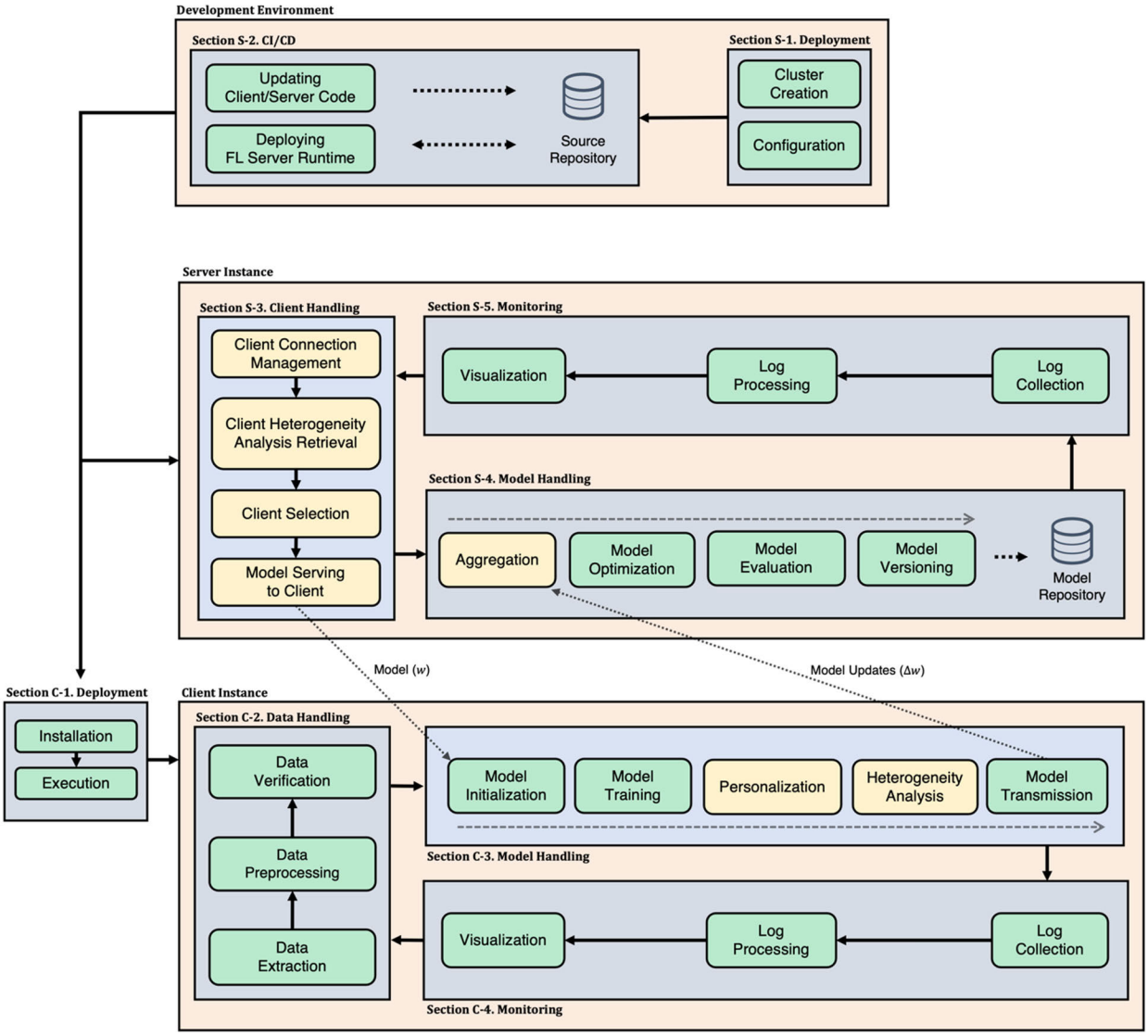

FIGURE 3. Overview on FL lifecycles in FedOps. It is largely distinguished into the server and client lifecycle, and sections according to the function constitute each server. The key functions for the FL process were highlighted.

ML libraries like PyTorch or TensorFlow in Python runtime is restricted. So they should be distributed by porting as an exclusive runtime environment [7]. Despite having the same instruction sets, multiple types of distributions are required considering the system constraints such as the presence of the graphics processing units (GPU) acceleration or memory capacity. In *FedOps Platform*, we have constructed various containerized environments for heterogeneous systems. This approach allows researchers to focus on their FL tasks without concerns about system constraints.

2) DATA HANDLING

In the data handling pipeline, operations such as data extraction, preprocessing, and validation are performed. In FL, considering that data is not transferred to the server, this pipeline is presented only in the client lifecycle. In MLOps, the data processing pipeline can be reorganized based on the decision of data scientist. But in *FedOps*, modifying the data processing logic of the client after the distribution of the application is difficult, therefore, updating the client should proceed with carefully tested code. Additionally, it should be automated to allow data preprocessing and validation considering the task to be undertaken in advance.

3) MODEL HANDLING

In the model handling pipeline, a client device trains local model and serves it according to each step, proceeding as follows:

(i) Model initialization: global model of the initial state is transferred via communication with the server. Here, pretrained model is ideal to initialize since it allows fast optimization for the local model.

(ii) Model training: local model trains using local data from the client device. Here, convergence speed by training is affected by device performance, and it also can differ based on the extent of personalization and fine-tuning.

FIGURE 4. Real-time monitoring dashboard of client generated owing to the log processing pipeline. Local model and data were visualized on the webpage.

(iii) Personalization: this process is an operation for personalizing the global model suitable for the client. It can be especially useful in training paradigms like transfer learning and multi-task learning. And personalization can be achieved using ML techniques like meta-learning and adding a personalization layer [25].

(iv) Heterogeneity analysis: an independent test is proceeded to measure the network bandwidth and systems heterogeneity of the client devices. The analysis results will be used to address performance degradation factors of FL. We performed the benchmark test for the client using proposed *chunk-bench* method. Details on this method are shown in Section IV.

(v) Model transmission: this process is transferring the weight parameter updates that are the results of local model optimization to the server for global model aggregation.

In *FedOps Platform*, Apache Airflow [26] was used to constitute the model serving pipeline. Using this workflow management tool, each task in the pipeline is represented as a node of directed acyclic graph (DAG) to enable sequential execution, thereby indicating success or failure as a result.

4) MONITORING

In the FL environment, tracking the client log into the server corresponds to an invasion of privacy, therefore, monitoring the client is considered to be restricted. However, if it is an application that can provide statistical insights and benefits to the user and not the server, there still remain reasons to implement client-side monitoring. For example, for a healthcare application that collects vital sign data by communicating with the sensor, an alert that visualizes the current user's health state can be shown to the user when in emergency.

In the *FedOps Platform*, client monitoring is strictly distinguished from the server and passed through an independent data processing pipeline. In Fig. 4, client data processing was facilitated by using the ELK stack [27], which comprises Logstash, Elasticsearch, and Kibana. Detailed information on the data processing pipeline is provided in Section III-C below.

C. SERVER LIFECYCLE

As shown in Sections S-1 to S-5 of Fig. 3, server manages the client state, updates the global model from local model weights of the clients, and performs the core function of monitoring as a primary part in FL. *FedOps Platform* can be easily distributed either to on-premise or cloud environment, and is enabled to efficiently allocate computing resources through clustering of available servers.

The function and composition of submodules are as below:

1) DEPLOYMENT

The deployment process in the server begins by composing the cluster from server nodes in which FL service resources can be allocated. The cluster is generally composed of Kubernetes which supports the container environment, multi-GPU, load balancer, and service mesh configurations. They are effective in distributing the load of the cluster owing to the efficient system resource allocation and multiuser access.

As a real case, we used MicroK8s [28] as one of the Kubernetes distros from the *FedOps Platform*, Ingress-NGINX [29], MetalLB [30] as the load balancer and Istiod [31] as the service mesh. *FedOps Platform* that includes such cloud-native applications can be deployed either in on-premise environments or the cloud using package managers like Helm chart [32] and Juju [33].

2) CI/CD

Client and server code can be updated as development proceeds, and the runtime based on the code update can be deployed for regular service. Before the deployment, updates based on the insight from monitoring previous FL lifecycle should be reflected.

In application deployment, we implemented real-time synchronization with the application via remote connection of Git repository and automation using a GitOps CI/CD tool such as ArgoCD [34].

3) CLIENT HANDLING

In FL, connection of the server with the current clients must be maintained before and ongoing training, the state of current clients should be detected by server, and the server state should be broadcasted to the clients. The key processes are as follows:

(i) Client connection management: server manages connected clients and keeps connection until the whole FL cycle ends. This operation could proceed with interfacing client-side application. Server identifies the availability of model training by updating the client states such as the network status (e.g., unmetered Wi-Fi connection), battery life, and idleness [24]. Even during ongoing model training, it is imperative to handle exceptional cases, for example, by attempting reconnection in the event of a client's lost connection. We used gRPC to communicate with the client and FastAPI to communicate within the internal cluster. A client manager that broadcasts the client state was constructed with functions such as health check or transmitting signal that starts local model training.

(ii) Client heterogeneity analysis retrieval: *FedOps Platform* uses client metadata, which is obtained by heterogeneity analysis through benchmark test from the clients. This metadata can be used to improve the user experience by updating client evaluation. We analyzed systems heterogeneity of clients using the *chunk-bench*. Details on *chunk-bench* can be found in Section IV.

(iii) Client selection: clients that participate in model training are selected based on a certain threshold. As a result of client selection, convergence speed of selected local models can be accelerated and communication cost can be reduced. When this operation finishes, the server requests model training to the clients, and the clients start local model training.

(iv) Model serving to client: global model within the server is broadcasted to the clients. Here, the global model in the server is loaded from a model repository using persistent storage.

4) MODEL HANDLING

In the global model handling pipeline, selecting an aggregation strategy is the core process, which is about how to calculate collected local model weights. The other processes of the model handling pipeline are identical to those in general MLOps.

Aggregation is an aggregating and computing operation from the result of local model training. The aggregation algorithm affects the global model performance, and this implies effect on local model performance again after one lifecycle.

We used Flower [5] as the core engine for handling FL model and transferring model updates. And PyTorch was used to evaluate the global model and to train the local model.

5) MONITORING

In FL, since data is not centralized, monitoring clients from the server is limited compared to typical ML services. However, without invading privacy, client metrics and metadata can be effective monitoring resources that provide insights to the server from the clients. In addition, monitoring FL-specific circumstances can elicit immediate performance feedback by visualizing communication cost, comparison of various aggregation algorithms in *FedOps*.

To implement the monitoring module, we collaboratively used several applications that could constitute a log processing pipeline in the cluster. To collect and transmit logs, Logstash was used in log parsing, Elasticsearch was used in log storing and indexing, and Kibana was used for visualization [27]. Consequently, local and global model data can be monitored in real-time, as shown in Fig. 5. Additionally, weights and biases [35] was used for hyperparameter tuning and monitoring the progress of model training. Fig. 7 shows that Weavescope [36] was used for real-time network dependencies visualization between Kubernetes resources and other external components.

D. HETEROGENEITY ANALYSIS

Addressing heterogeneous clients that could harm the model performance and impede FL model training process is one of the key challenges that should be covered in *FedOps*. We propose *chunk-bench* tool and demonstrate a client selection method with systems heterogeneity analysis strategy.

1) HETEROGENEITY BENCHMARK

Chunk-bench analyzes the communication cost and systems heterogeneity of the client in the *FedOps Platform*, and it provides insights to improve performance in the next FL cycle. *Chunk-bench* uses only a small *chunk* of the data so that the clients can terminate the test in the shortest time possible. The benchmark tracks and records the systems performance of all clients. Subsequently, it visualizes the communication cost and the time-to-accuracy of each client to provide insights to FL server.

For the experiment, we set the data ratio for the *chunk-bench* to 50% of batch data as we use lightweight data for training. If a bigger model or data is used, the *chunk-bench* can be performed with a smaller amount of data. When we reduced the amount of data by 50%, the average local model training time of the initial communication round for

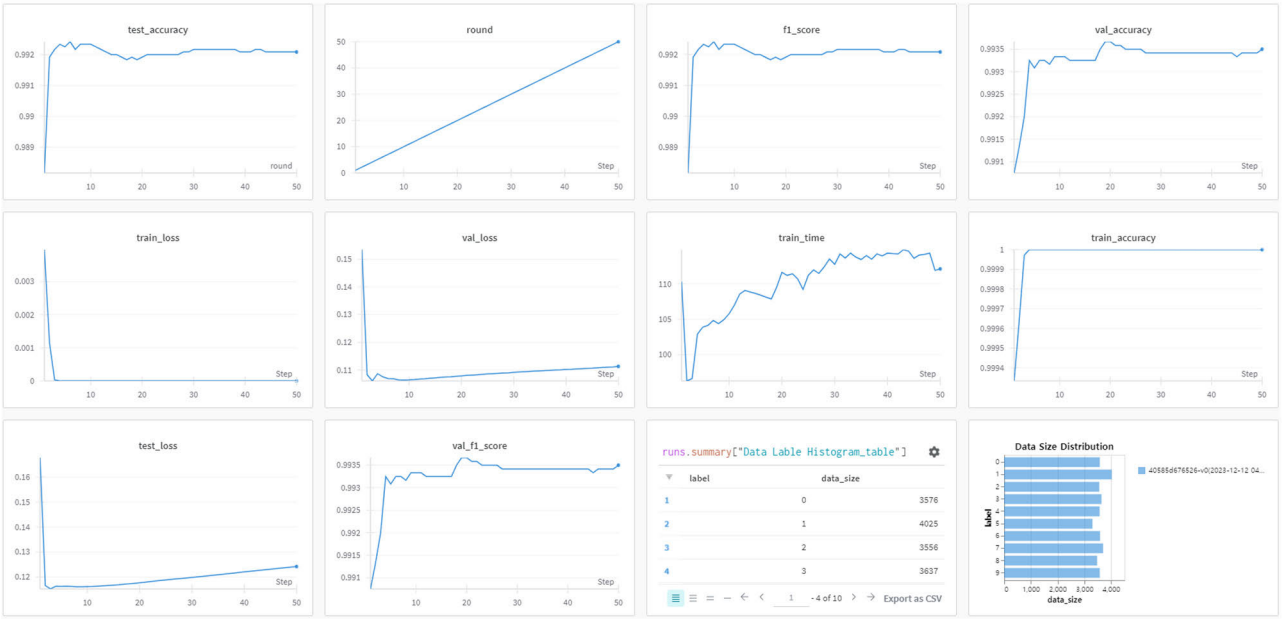

FIGURE 5. Real-time Monitoring dashboard of FedOps Platform generated by the log processing pipeline. Local and global model data were visualized on the webpage.

50 clients decreased by 54.4%. Details on the settings of *chunk-bench* for model training and FL clients are presented in Section IV-A.

Fig. 6 demonstrates *chunk-bench* to quantify and identify the system heterogeneity among FL clients. The client numbers were assigned sequentially according to the order in which clients submitted their model training outcomes to the server following the commencement of the communication round. We synchronized the system clock to a network time protocol (NTP) server identical to the client and server at the start of the measurement, ensuring time reliability. Here, it is important to note that there could be varying patterns in the network latency of the client during real-world device communication, as we conducted communication between clients deployed within the internal cluster.

The elapsed time of clients in a communication round comprises the communication time and training time as shown in the chart. The communication time is mainly determined by the communication speed of the clients and the bandwidth of the server network. And the training time is mainly determined by ML performance of the local devices.

Fig. 6 also shows the communication time of a few clients can be 300 times longer than the time of other clients. The corresponding clients were slowly aggregated owing to the communication time despite having already completed the local training round. The cause of these clients is assumed to be many clients communicating with a server simultaneously, resulting in server-side network overhead. The server-side network overhead is significantly higher in the communication time before training than after training. This discrepancy is due to less communication being sent per unit time to the server after training, as the completion time varies among FL clients due to differences in their model training performance.

Based on the communication results, we calculated average overhead time. The average overhead time, an indicator of the server's ability to handle concurrent FL clients, is determined by averaging the values for which the communication time surpasses 0.1 seconds as per the *chunk-bench* outcome. The average overhead time after three experimental measurements was 2.13 ± 0.20 (s), and the average number of clients that experienced the overhead was 20.33 ± 0.5. This indicates that out of 50 clients, 20 clients on average experienced a communication delay of 2 seconds on average. Here, the communication delay mainly occurs owing to the server-side network overhead in the experiments, but it could vary based on the network connection state of the client. To reduce this server-side network overhead, *FedOps Platform* has an option that limits the number of simultaneous communications per unit of time through client scheduling [24].

Although we focus on systems heterogeneity in this section, *FedOps Platform* provides information to address data heterogeneity by analyzing the loss of clients. Fig. 8 illustrates the log-scaled loss of clients across communication rounds, offering FL researchers loss metric for client evaluation and selection. Client loss information applied to proximal terms can help addressing data heterogeneity. By evaluating the loss information, one can dynamically adjust the strength of proximal terms. This allows the system to delicately balance between catering to individual client needs (due to data heterogeneity) and maintaining overall model coherence. Hence, using loss as a guide, proximal regularization can be effectively wielded to address inherent

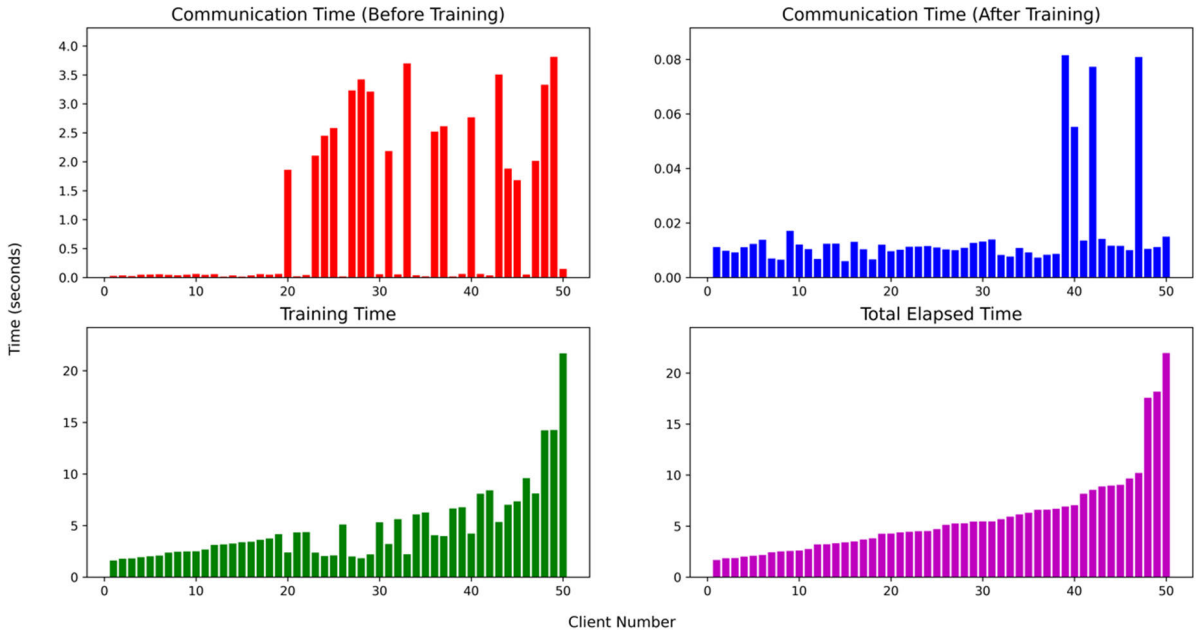

FIGURE 6. Performance time distribution for each client in the first FL communication round during chunk-bench. This figure is distinguished into four sub-charts and presented into the communication time before training and after training, training time, and the total time spent. The client number was arranged in ascending order of the fastest client response time.

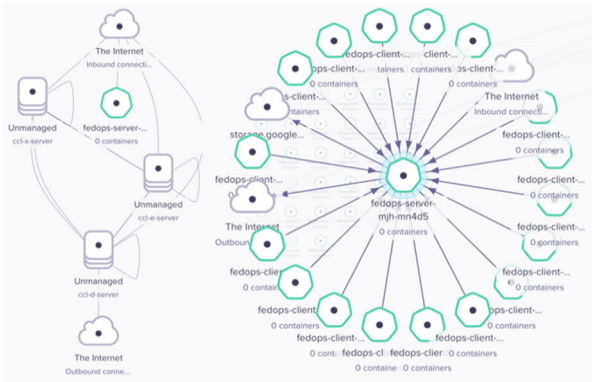

FIGURE 7. Server-side network topology which visualizes connections with clients.

data heterogeneity across clients in a federated setting. Here, loss implies the potential for further local model optimization [14].

2) CLIENT SELECTION

We identified an imbalance in local models training owing to systems heterogeneity. This imbalance implies that a few clients with slow model training speed can consume the system resources of most other clients, which is a vicious cycle that could continue with each communication round. To address this inefficiency, we performed client selection using systems heterogeneity. To present an example of client selection, we propose a metric as a client adaptive threshold (*CAT*) that is adaptively determined based on the *chunk-bench* result. Clients that exceed the threshold will be excluded from model training in the subsequent communication rounds. The equation for *CAT* is as follows:

$$\begin{cases} \text{if } n \text{ is odd}, & median(T_c) = T_{\left[\frac{n+1}{2}\right]} \\ \text{if } n \text{ is even}, & median(T_c) = \frac{1}{2} \cdot \left(T_{\left[\frac{n}{2}\right]} + T_{\left[\frac{n}{2}+1\right]}\right) \end{cases}$$

For a given set T_c of univariate values representing *chunk-bench* execution times $T_1, T_2, T_3, \ldots, T_n$ for clients derived from n clients, the median is defined as shown above.

$$MAD = median(|T_i - median(T_c)|)$$

We used median absolute deviation (*MAD*) to determine client drop policy for each section by presenting robust deviation for the clients with abnormally slow *chunk-bench* execution [37]. The *MAD* obtains the median value again from the deviation that subtracted the median of T_c from the client execution time.

$$CAT = median(T_c) + k \cdot MAD$$

CAT can be obtained using the *MAD* value and adjusting the *MAD* coefficient k, which is a hyperparameter to moderate the threshold. As the k value increases, the *CAT* value increases as well, such that the clients with lower values are excluded. For the policy that is reluctant to drop clients, k can be adjusted upward to increase the threshold.

IV. EVALUATION

We aim to evaluate *FedOps Platform* if the platform can mitigate performance degradation in FL by analyzing systems heterogeneity of clients. The primary objective of this experiment is to improve the performance of FL process by enhancing time-to-accuracy and reducing the communication cost.

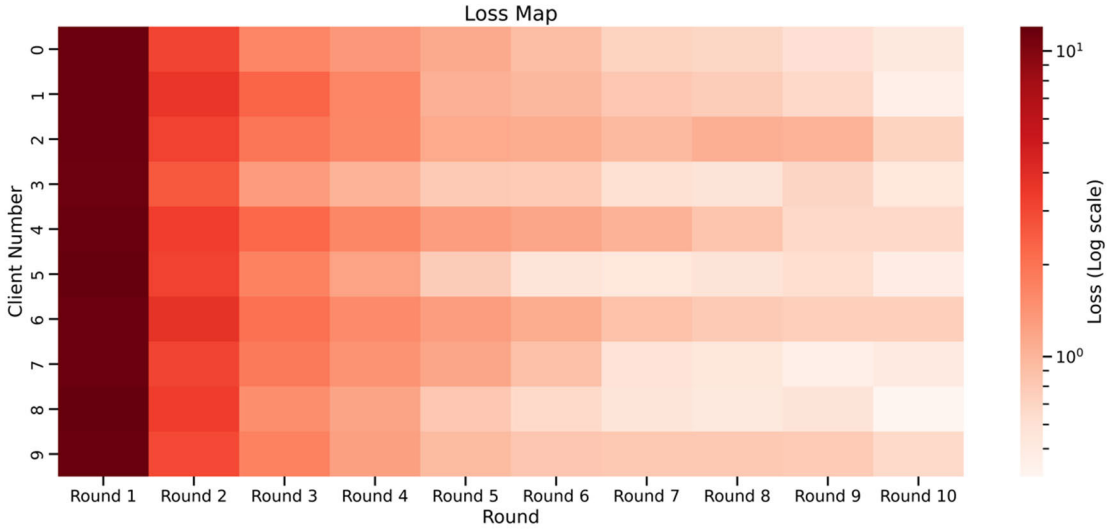

FIGURE 8. Log-scaled loss map of clients. Darker area indicates that client has higher loss in the result of communication round.

A. METHODOLOGY

1) CLIENT SIMULATION

To analyze systems heterogeneity and create a client according to performance distribution, a simulation of FL clients was conducted in containerized environments. We used resource allocation of Kubernetes to differentiate the clients based on their performance through differentiating the number of central processing unit (CPU) cores of FL clients. We identified in advance through normal multithreading that the performance improved as the number of CPU cores increased [38]. Here, the performance for each CPU core of each server used in the experiment varies.

In the test process, we encountered an issue with PyTorch within the container environment, which excessively slowed down the model training. We determined that this slowdown was likely due to problems related to CPU scheduling. This problem was resolved by limiting the number of CPU cores to double the target CPU cores by calling *torch.set_num_threads (int: cores)* function.

To simulate the realistic FL clients, we utilized Smartphone Processor's Scores data [39]. This data is extracted from the ML benchmark results of the mobile benchmarking tool, Geekbench, which scores the mobile processor ML performance of 188 smartphones released until 2022. The result values of each device were averaged, and only the devices that performed a minimum of five tests were listed. Fig. 9 illustrates a ten-fold difference in performance between the highest and lowest scoring mobile CPUs. This shows that training time on local devices can vary dramatically, implying that most devices will remain idle in most of the time during the communication rounds.

To simulate real-world systems heterogeneity with respect to mobile ML performance data, we transformed original CPU scores into equivalent CPU-core resource allocations. We achieved this by generating a probability distribution from the original data across 100 discrete bins. This histogram

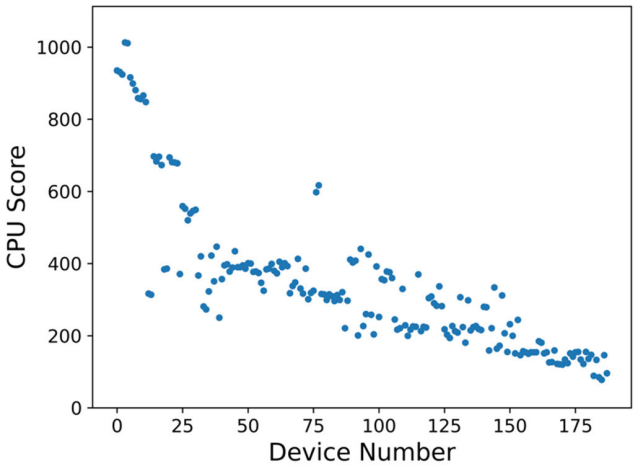

FIGURE 9. Mobile CPU score data distribution of ML benchmark test.

representation ensured that the performance characteristics of the CPUs were adequately captured in our simulation. We then adjusted the scale according to the number of CPU cores in the cluster and sampled the number of CPU cores for 50 clients based on the probability distribution of the histogram. We set the target CPU cores at 1,400 millicores (equivalent to 1.4 cores) to scale the original data. The scaling process resulted in an average CPU cores at 1,400 millicores, with a standard deviation at 821.59 millicores. Upon conducting five sets of sampling for 50 clients using the discrete distribution, the average CPU resource of 50 clients was $1{,}401 \pm 53.27$ millicores, and the standard deviation was 837 ± 99.41 millicores. Note that processing performance of FL task is not directly proportional to the number of CPU cores.

2) EXPERIMENTAL SETUP

In experimental setup, client and server runtime were implemented in the containerized environment to create

TABLE 2. Hardware specifications for experiments.

Server	CPU	RAM	Operating System	Software Dependencies
Server-α	Intel Core i9-9900KF @ 3.60GHz (8 cores, 16 threads)	64GB DDR4-2666	Ubuntu 20.04 LTS	Kubernetes v1.22
Server-β	2x Intel Xeon Silver 4214R @ 2.40GHz (24 cores, 48 threads in total)	64GB DDR4-3200	Ubuntu 20.04 LTS	Kubernetes v1.22
Server-γ	Intel Core i9-10980XE @ 3.00GHz (18 cores, 36 threads)	64GB DDR4-2133	Ubuntu 20.04 LTS	Kubernetes v1.22

scalable form which is adaptable to the cloud. The container deployment was performed as a Job which is a resource type of Kubernetes. Job automatically performs arrangement for a certain task and returns either a success or failure as the execution result of the deployed Pod. Pod, as the Kubernetes resource type, simulates clients by running the containers with pre-defined performance limits. We used resource allocation policy on Pod to identically fix the requests and limits of the CPU core and set the hard limit for maintaining performance to reduce variability in experimental environments. Table 2 shows a cluster configuration, constituting three servers, with CPU resources in Kubernetes allocated dynamically.

We used Python version 3.9, PyTorch version 1.13, and Flower version 1.3 to implement the experiment. Flower is an open-source FL framework that offers various FL algorithm baselines [14]. With a real device overhead of less than 100ms per communication round and the ability to scale to 15M clients, Flower has the ideal specifications to implement *FedOps Platform*. To manage the connection between FL participants, on the top of Flower, we added components such as client manager and server manager, with the functions for analyzing the client heterogeneity and executing the client drop policy.

3) DATASET AND MODEL

We used CIFAR-10 [40] dataset was used for the experiments. CIFAR-10 is a dataset for image classification purposes, and it has 50,000 train and 10,000 test image data points that are classified into 10 classes of objects. We also used MNIST dataset [41], which consists 60,000 train and 10,000 test image data points that are classified into 10 classes of handwritten digit numbers. To train different local data for 50 clients, we partitioned the data into 50 subsets.

For image classification, we experimented with a basic convolutional neural network (CNN) model, FedAvg [5] and FedProx [6] as the aggregation algorithm. For the baseline CNN model, we set the local mini-batch size to 64, local epochs to 5, the learning rate to 0.001, and the momentum to 0.9. We used stochastic gradient descent (SGD) as an optimizer. And we used a basic logistic regression model which is shown in FedProx paper to classify MNIST dataset. We also followed the hyperparameters setting along the original FedProx paper.

B. RESULTS

1) EVALUATION METRICS

In the discussion in Section III-D, we proposed *chunk-bench*, which is systems heterogeneity analysis tool and *CAT* as a sample client selection criteria. To verify *chunk-bench* and *CAT*, we measured time-to-accuracy performance, model accuracy, and additional communication cost metrics. We proceeded with three experiments for each performance simulation, and the average and standard deviation of experimental results will be listed as evaluation metrics.

2) EVALUATION RESULTS

Table 3 and Table 4 show the comparison results of baseline FedAvg [5] and proposed *CAT* methods. Loss and accuracy are the metrics of measurement results at 500 communication rounds where the global model converges. Elapsed time is the time required to train 500 communication rounds, and the time-to-accuracy compares the relative time required to achieve a corresponding accuracy at 500 communication round.

The experiment showed that the *CAT* improved model training time without sacrificing accuracy and loss in $k = 5$ and $k = 7$. As elapsed time decreased, the rate of clients dropping was most significant at 87.7% faster than the baseline FedAvg when $k = 3$, and 37.8% faster when $k = 5$ in CIFAR-10 dataset. For the time-to-accuracy, 17–41% time reduction from the baseline. This improvement in the model training time exceeded the rate of data loss, which was the rate of dropped clients due to client selection. Detailed information on dropped clients is summarized in Table 5.

Table 5 shows the measured communication indices from the server process in the experiment using CIFAR-10 dataset. It details the transmitted and received network traffic from the server to the client up to 500 communication rounds. The communication cost-to-accuracy refers to the total communication that the server transmitted and received to achieve the accuracy of 80%. The dropped client proportion expresses the rate of clients that were dropped due to *CAT* as the FL participation cut-off.

The transmitted traffic from the *FedOps Platform* to the server was reduced by 5–20% compared to the baseline FedAvg. The received traffic was reduced by up to 15%. Additionally, the communication cost for achieving a certain accuracy was reduced by 11–30%, which was greater than the dropped client's proportion. This result shows that *FedOps Platform* can improve the communication

TABLE 3. Evaluation results on CIFAR-10 datasets.

Dataset	CIFAR-10			
Metrics	Loss	Accuracy	Elapsed Time (min.)	Time-to-Accuracy
FedAvg	3.86 ± 0.08	.832 ± .003	153. ± 14.5	131. ± 11.8 (x1.68)
FedAvg + *CAT* ($k = 3$)	5.97 ± .105	.806 ± .016	**81.5 ± 3.86**	**77.8 ± 12.8 (x1.00)**
FedAvg + *CAT* ($k = 5$)	3.55 ± 0.24	**.844 ± .010**	111. ± 5.58	83.4 ± 9.64 (x1.07)
FedAvg + *CAT* ($k = 7$)	**3.50 ± 0.10**	.842 ± .002	141. ± 23.2	109. ± 18.8 (x1.40)
FedProx	**1.35 ± .025**	**.790 ± .009**	305. ± .424	387. ± 3.97 (x1.84)
FedProx + *CAT* ($k = 3$)	2.19 ± .005	.736 ± .002	**154. ± 13.6**	**210. ± 18.2 (x1.00)**
FedProx + *CAT* ($k = 5$)	1.62 ± .100	.776 ± .037	181. ± 17.4	232. ± 12.8 (x1.11)
FedProx + *CAT* ($k = 7$)	1.44 ± .035	.783 ± .027	192. ± 4.52	245. ± 13.6 (x1.17)

TABLE 4. Evaluation results on MNIST datasets.

Dataset	MNIST			
Metrics	Loss	Accuracy	Elapsed Time (min.)	Time-to-Accuracy
FedAvg	.092 ± .015	.658 ± .075	306. ± 5.65	470. ± 65.8 (x1.94)
FedAvg + *CAT* ($k = 3$)	.096 ± .019	.665 ± .051	**159. ± 44.2**	**242. ± 84.5** (x1.00)
FedAvg + *CAT* ($k = 5$)	**.078 ± .015**	**.737 ± .025**	195. ± 78.7	266. ± 113. (x1.10)
FedAvg + *CAT* ($k = 7$)	.106 ± .035	.616 ± .153	259. ± 9.11	438. ± 99.8 (x1.80)
FedProx	.049 ± .046	.848 ± .020	560. ± 173.	658. ± 197. (x2.63)
FedProx + *CAT* ($k = 3$)	.050 ± .003	.836 ± .013	**209. ± 20.2**	**250. ± 25.3 (x1.00)**
FedProx + *CAT* ($k = 5$)	.049 ± .004	.838 ± .011	230. ± 73.0	274. ± 86.3 (x1.10)
FedProx + *CAT* ($k = 7$)	**.042 ± .040**	**.872 ± .008**	296. ± 36.9	340. ± 40.5 (x1.36)

TABLE 5. Measured communication cost and relevant indices of experiments.

Methods	FedAvg	FedAvg + *CAT* ($k = 3$)	FedAvg + *CAT* ($k = 5$)	FedAvg + *CAT* ($k = 7$)
Transmitted Traffic (GB)	11.7 ± 0.01 (x1.00)	**9.37 ± 0.80 (x0.80)**	10.5 ± 0.45 (x0.90)	11.1 ± 0.15 (x0.95)
Received Traffic (GB)	5.66 ± 0.55 (x1.00)	**4.81 ± 0.41 (x0.85)**	5.37 ± 0.23 (x0.95)	5.69 ± 0.08 (x1.01)
Communication Cost to Accuracy (GB)	14.3 ± 1.38 (x1.00)	**10.0 ± 1.15 (x0.70)**	11.7 ± 1.30 (x0.82)	12.7 ± 0.31 (x0.89)
Dropped Clients Proportion	**N/A**	0.19 ± 0.05 (x1.00)	0.10 ± 0.04 (x0.53)	0.05 ± 0.01 (x0.26)

cost-to-accuracy through *CAT* to support communication-efficient FL.

Fig. 10 and Fig. 11 show the evaluation metrics graph of each method according to the communication rounds during the CIFAR-10 data training process. Based on the communication rounds, the *CAT* methods resulted in faster convergence speed. What is notable from this graph is, for $k = 5$ and $k = 7$, which dropped relatively fewer clients, both time-to-accuracy and communication cost-to-accuracy were reduced without any accuracy degradation in the results.

However, Fig. 11 shows that for $k = 3$ that dropped the most clients among experiment methods, loss function did not converge after sufficient communication rounds. This implies that *CAT* and time-to-accuracy are in a trade-off relationship in system heterogenous setting, which means when the rate of client drop increases over a certain level, significant accuracy degradation occurs.

C. DISCUSSION

The experiments presented an example of client selection improved time-to-accuracy and communication cost with minimal accuracy loss. FL service will not waste client device resources, so user experience of the service can be enhanced. This shows *chunk-bench* offers insights into client heterogeneity, and engineers can fully utilize the customizable client selection feature to implement their own methods.

However, it should be considered that the setting in the experiment has the potential risk of creating a biased model due to connectivity conditions of the user. For example, relatively more clients might be selected from buildings, cities, or nations with high network bandwidth.

Note that as we focused on systems heterogeneity, data heterogeneity was not covered in the experiment. In actual client selection scenario, this should be considered. As we focused

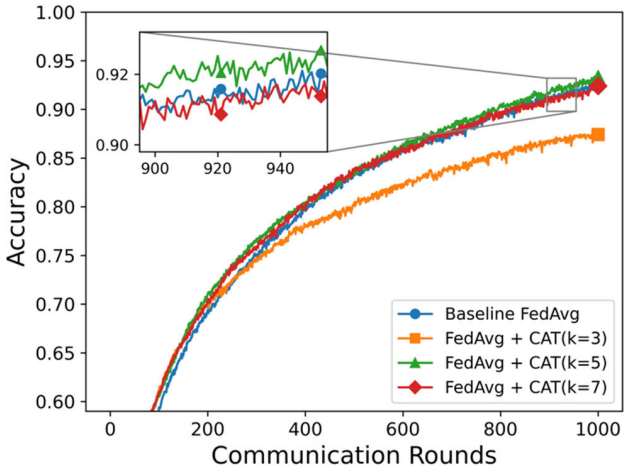

FIGURE 10. Accuracy resulting from performance evaluation and CAT value adjustment.

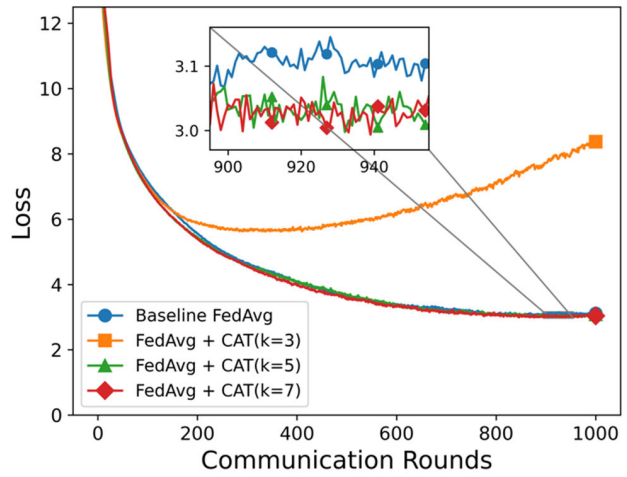

FIGURE 11. Loss resulting from performance evaluation and CAT value adjustment.

on showing the functionality of *FedOps Platform*, we did not use advanced aggregation strategy and personalization algorithm. But *chunk-bench* can be used with other non-IID data handling methods or client selection policy, therefore, a greater improvement in performance is expected on *FedOps Platform* in the future.

V. CONCLUSION

In this study, we demonstrated the system architecture and best practices of *FedOps* and presented the implementation for *FedOps Platform*. *FedOps Platform* supports various aggregation algorithms, edge devices, and simulation environments. *FedOps Platform* enables researchers to enhance their methods and promote engineers to develop communication-aware FL applications as real-world services. We expect *FedOps Platform* to significantly simplify construction and management end-to-end FL projects.

Although *FedOps Platform* supports various FL methods and runtime environments, it has yet uncovered areas in FL research. We have not introduced advanced personalization and optimization techniques to our platform. As future work, there can be applied various client selection algorithms, blockchain FL with a totally decentralized network and integration with decentralized large language models.

Finally, we showed *chunk-bench* method, facilitating the analysis of heterogeneous clients through containerized client simulation. We analyzed the communication cost to demonstrate the feasibility of reducing the communication cost-to-accuracy between clients and server. *FedOps Platform* offers an optimal solution for the end-to-end development of FL projects, with improving both computational and communication efficiencies.

REFERENCES

[1] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2019.

[2] A. Yang, Z. Ma, C. Zhang, Y. Han, Z. Hu, W. Zhang, X. Huang, and Y. Wu, "Review on application progress of federated learning model and security hazard protection," *Digit. Commun. Netw.*, vol. 9, no. 1, pp. 146–158, Feb. 2023.

[3] M. A. P. Chamikara, P. Bertok, I. Khalil, D. Liu, and S. Camtepe, "Privacy preserving distributed machine learning with federated learning," *Comput. Commun.*, vol. 171, pp. 112–125, Apr. 2021.

[4] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–17, 2022.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artif. Intell. Statist.*, 2017.

[6] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith "Federated optimization in heterogeneous networks," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 429–450, 2018.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017.

[8] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*.

[9] Y. Jiang, J. Konečný, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," 2019, *arXiv:1909.12488*.

[10] J. Hamer, M. Mohri, and A. T. Suresh, "FedBoost: A communication-efficient algorithm for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2020.

[11] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, "FedML: A research library and benchmark for federated machine learning," 2020, *arXiv:2007.13518*.

[12] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "FATE: An industrial grade platform for collaborative learning with data protection," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 1–6, 2021.

[13] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

[14] J. Shin, Y. Li, Y. Liu, and S.-J. Lee, "FedBalancer: Data and pace control for efficient federated learning on heterogeneous clients," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Services*, Jun. 2022.

[15] 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*.

[16] L. Li, J. Wang, and C. Xu, "FLSim: An extensible and reusable simulation framework for federated learning," in *Proc. Int. Conf. Simulation Tools Techn.* Cham, Switzerland: Springer, 2020, pp. 350–369.

[17] A. Ingerman and K. Ostrowski. (2019). *Introducing TensorFlow Federated*. Accessed: Mar. 26, 2023. [Online]. Available: https://blog.tensorflow.org/2019/03/introducing-tensorflow-federated.html

[18] M. H. Garcia, A. Manoel, D. M. Diaz, F. Mireshghallah, R. Sim, and D. Dimitriadis, "FLUTE: A scalable, extensible framework for high-performance federated learning simulations," 2022, *arXiv:2203.13789*.

[19] Y. Wen, W. Li, H. Roth, and P. Dogra. *Federated Learning Powered by NVIDIA Clara*. Accessed: Aug. 23, 2023. [Online]. Available: https://developer.nvidia.com/blog/federated-learning-clara/
[20] Google Cloud. (Mar. 24, 2023). *MLOps: Continuous Delivery and Automation Pipelines in Machine Learning*. Accessed: Mar. 26, 2023. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning
[21] Microsoft. (Sep. 23, 2022). *Machine Learning Operations*. Accessed: Mar. 26, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/ai-machine-learning-mlops
[22] Iguazio. *Data Science and Machine-Learning Operations (MLOps)*. Accessed: Mar. 26, 2023. [Online]. Available: https://www.iguazio.com/docs/latest-release/ds-and-mlops/
[23] S. Yang, J. Moon, J. Kim, K. Lee, and K. Lee, "FLScalize: Federated learning lifecycle management platform," *IEEE Access*, vol. 11, pp. 47212–47222, 2023.
[24] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 374–388.
[25] V. Kulkarni, M. Kulkarni, and A. Pant, "Survey of personalization techniques for federated learning," in *Proc. 4th World Conf. Smart Trends Syst., Secur. Sustainability (WorldS)*, Jul. 2020.
[26] The Apache Software Foundation. *Apache Airflow*. Accessed: Aug. 23, 2023. [Online]. Available: https://airflow.apache.org/
[27] Elastic N.V. *Elastic Stack: Elasticsearch, Kibana, Beats, and Logstash*. Accessed: Mar. 26, 2023. [Online]. Available: https://www.elastic.co/kr/elastic-stack/
[28] Canonical Ltd. *MicroK8s—Zero-Ops Kubernetes for Developers, Edge and IoT*. Accessed: Mar. 26, 2023. [Online]. Available: https://microk8s.io/
[29] NGINX. *NGINX Ingress Controller*. Accessed: Mar. 26, 2023. [Online]. Available: https://docs.nginx.com/nginx-ingress-controller/
[30] MetalLB. *MetalLB, Bare Metal Load-Balancer for Kubernetes*. Accessed: Mar. 26, 2023. [Online]. Available: https://metallb.universe.tf/
[31] Istio. *Istio*. Accessed: Mar. 26, 2023. [Online]. Available: https://istio.io/
[32] Helm. *Helm*. Accessed: Mar. 26, 2023. [Online]. Available: https://helm.sh/
[33] Canonical Ltd. *Juju—The Simplest Way to Deploy and Maintain Applications in the Cloud*. Accessed: Mar. 26, 2023. [Online]. Available: https://juju.is/
[34] Argo CD. *Argo CD—Declarative GitOps CD for Kubernetes*. Accessed: Mar. 26, 2023. [Online]. Available: https://argo-cd.readthedocs.io/en/stable/
[35] Weights & Biases. *Weights & Biases—Developer tools for ML*. Accessed: Mar. 26, 2023. [Online]. Available: https://wandb.ai/site
[36] Weaveworks Inc. *Weave Scope: Automatically Detect and Process Containers and Hosts*. Accessed: Mar. 26, 2023. [Online]. Available: https://www.weave.works/oss/scope/
[37] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *J. Exp. Social Psychol.*, vol. 49, no. 4, pp. 764–766, Jul. 2013.
[38] PyTorch. *CPU Threading and TorchScript Inference*. Accessed: Mar. 26, 2023. [Online]. Available: https://pytorch.org/docs/stable/notes/cpu_threading_torchscript_inference.html
[39] A. Zhou. (Mar. 2022). *Smartphone Processors Ranking & Scores, V1.0*. Accessed: Mar. 26, 2023. [Online]. Available: https://www.kaggle.com/datasets/alanjo/smartphone-processors-ranking
[40] A. Krizhevsky et al. *The CIFAR-10 Dataset*. Accessed: Aug. 23, 2023. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[41] Y. Lecun, C. Cortes, and C. J. Burges, "MNIST handwritten digit database," ATT Labs., Tech. Rep., 2010, vol. 2. [Online]. Available: http://yann.lecun.com/exdb/mnist

JIHWAN MOON is currently pursuing the B.S. degree in computer engineering with the Department of Computer Engineering, College of IT Convergence, Gachon University, Seongnam-si, Republic of Korea.

Since 2020, he has been a Researcher with the Cognitive Computing Laboratory, Gachon Institute of Artificial Intelligence. His current research interests include federated learning, reinforcement learning, generative model, automation, multi-modal analysis, and MLOps.

SEMO YANG received the B.S. degree in computer engineering, in 2022. He is currently pursuing the M.S. degree in IT convergence engineering with the College of IT Convergence, Gachon University, Seongnam-si, Republic of Korea.

From 2022 to 2023, he was a Researcher with the Cognitive Computing Laboratory, Gachon Institute of Artificial Intelligence. His research interests include artificial intelligence, deep learning, federated learning, machine learning, and MLOps.

KANGYOON LEE (Member, IEEE) received the B.S. degree in electronics engineering and the M.S. degree in computer science from Yonsei University, Seoul, South Korea, in 1986 and 1996, respectively, and the Ph.D. degree in IT policy management from Soongsil University, Seoul, in 2010.

From 2008 to 2014, he was the Director of the IBM Korea Laboratory for the Ubiquitous Computing and Software Solutions Lab, and he was promoted to the Leader of the IBM Watson Business Unit, South Korea, in 2014. Since 2016, he has been a Professor with the Computer Engineering Department, College of IT Convergence, Gachon University, Seongnam-si, Republic of Korea. He has been serving as the Director of the Gachon Institute of Artificial Intelligence, since 2016. His research interests include cognitive computing, healthcare advising, the IoT platforms, and industry transformation.