IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 March 2023, accepted 1 May 2023, date of publication 11 May 2023, date of current version 18 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275439

RESEARCH ARTICLE

FLScalize: Federated Learning Lifecycle Management Platform

SEMO YANG[1], JIHWAN MOON[1], JINSOO KIM[2], KWANGKEE LEE[2], AND KANGYOON LEE[1], (Member, IEEE)

[1]Department of Computer Engineering, Gachon University, Seongnam-si 13120, South Korea
[2]Innopia Technologies Inc., Seongnam-si 13217, South Korea

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

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

ABSTRACT Federated learning (FL) that can train using machine learning methods without moving data have attracted interest owing to the focus on data privacy. Several FL platforms and frameworks are being developed with various open datasets. However, FL has not yet been fully utilized in real-world projects; instead, centralized ML models are still being used for AI. Since FL is composed of numerous clients and executed, it is necessary to manage the lifecycle such as model deployment and status management to multiple clients in order to operate FL. This study proposes FLScalize to enable AI researchers to apply their own custom data and models to FL environments and to deploy and manage the FL lifecycle. Researchers who develop these models should be able to easily and conveniently apply custom data and models developed in a centralized environment to FL environments, deploy and train multiple clients, and manage the lifecycle of the entire FL process. FLScalize can be used to simulate system heterogeneity and data heterogeneity, both of which are FL issues that occur in real FL environments. Furthermore, FLScalize provides a manager component that continuously manages the FL client and server required for real-world FL tasks and realizes an FL lifecycle management implementation that enables continuous integration, deployment, and training.

INDEX TERMS Federated learning, heterogeneous simulation, lifecycle management, platform.

I. INTRODUCTION

Federated learning is a machine learning (ML) technique in which multiple clients holding their own local data cooperate with each other under the management of a central server or service provider [1]. This enables performing local model-based learning with client-owned datasets without sharing data, thereby protecting the privacy of data producers and providers. Various frameworks and platform environments have been developed for FL, including Flower [2], FedScale [3], FATE [4], PyShift [5], and EasyFL [6]. Currently, studies are investigating the use of FL in fields such as medicine [7], imaging [8], industry [9], and natural language processing [10]. However, studies are mostly focusing on improving the performance of FL algorithms; meanwhile, studies on FL operation management for real-world tasks are lacking. Since FL involves a large number of clients and is executed across them, it is crucial to handle various aspects of its life cycle, including model deployment and status management, to ensure proper functioning of FL. In addition, because AI model researchers mainly study centralized ML AI, they need an environment in which they can easily study and apply FL. In this light, the present study proposes the FL operation to allow researchers to easily apply and deploy custom data and models and to manage their lifecycle in an FL environment. Existing MLOps provides functions for data management, model design, training, application, and deployment by setting up automation pipelines [11]. MLOps affords advantages such as continuous integration and deployment (CI/CD) of data and models that are applied

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

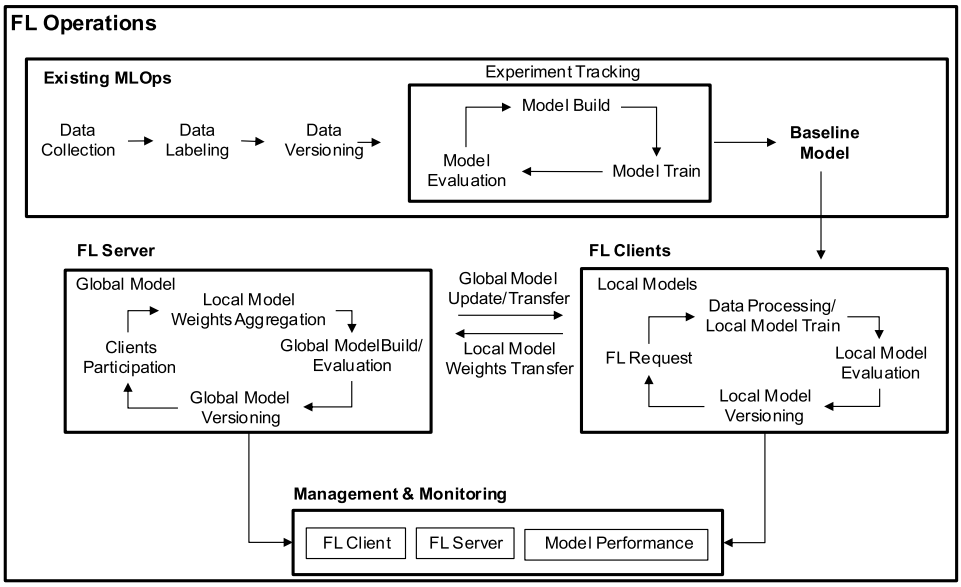


FIGURE 1. Configuration diagram of FL operation that manages the FL lifecycle by extending the existing MLOps concept.

in several fields [12]. MLOps only considers the automation pipeline for data, modeling, and deployment without considering the clients, the most important element of FL. In an FL environment, hundreds of thousands of clients can be configured, and therefore, data and models must be deployed to numerous clients. Furthermore, in an FL environment, the online/offline and training status of multiple clients and the server status must be managed, and lifecycle management is required to execute various FL tasks and to operate FL stably. As existing MLOps does not support these functions, it was extended to design and implement an FL operation, as shown in **Fig. 1**. Our FL operation concept is to manage the entire FL process of creating a global model by deploying and training the local model created in the experimental environment. **Table 1** shows a comparison between the FL operations and the existing MLOps.

This study proposes FLScalize for AI researchers who develop centralized models in FL environments. It allows the easy and convenient application of custom data and models, deployment across multiple clients, and lifecycle management.

TABLE 1. Comparison of existing MLOps and FL operations.

	Existing MLOps	FL Operations
Data Collect & Processing	○	○
Model Management	○	○
Multi Deployment	X	○
Client/Server Status Management	X	○

FLScalize makes the following contributions.

1. Easily application of data and models to the FL environment: Custom data and models can be easily applied to various FL task environments. The data and model application parts of FL are configured in a format similar to that of the centralized environment to provide an FL experimental and development environment for AI researchers.

2. Manager component that checks the status of FL client and server: A manager component that continuously checks and manages the status of the FL client and server is configured. Several FL clients need to identify the online/offline and training status to participate in FL rounds, and they must continuously manage them through the client manager. Likewise, the FL server must manage the online/offline and aggregation status through the server manager.

3. Continuous integration/deployment/training: For each of the various FL tasks, the latest versions of the FL client and server can be integrated and deployed easily, and FL rounds can be performed periodically.

4. Providing simulation environments for two categories of FL that occur in the real world: It is possible to simulate heterogeneous system and data, the two biggest issues in various FL tasks. System heterogeneity simulations can be performed by configuring the system resource environment differently for each client. In addition, each client can have different data by receiving an ID for each client, and data heterogeneity for IID and non-IID cases can be simulated by using the data partition function.

5. FL lifecycle management applicable to real projects: We provide the four functions mentioned above by supporting the simulation of FL tasks that can occur in the real world. FLScalize can manage an automated and reliable FL lifecycle.

TABLE 2. Comparison of FLScalize with existing FL frameworks and platforms. FLScalize can easily apply data and models and continuously manage FL lifecycle by adding manager component and CI/CD/CT functions.

	TFF	FATE	FedScale	EasyFL	FLScalize
Simple application of data and model	✓	✓	✓	○	○
FL client/server management	X	X	X	X	○
Heterogeneous system and data	✓	✓	✓	○	○
Multi-client CI/CD/CT	X	✓	X	X	○
FL lifecycle management	X	✓	X	✓	○

✓: Limited support

The rest of this paper is organized as follows. Section II introduces the FL research background, framework, and platform and reviews MLOps-related research. Section III presents the system architecture of FLScalize and the operation processes of all components. Section IV introduces the experimental environment of FLScalize. Section V presents the experimental results obtained with the experimental setup described in Section IV. Finally, Section VI summarizes this study and suggests future research directions.

II. RELATED RESEARCH

FL is a distributed learning method that allows multiple clients to train an ML model without moving data to the control of a central server, thereby ensuring data privacy [13]. Federated averaging (FedAvg) is the standard aggregation algorithm that is used most widely in FL. Ongoing FL studies have proposed FedProx [14] that considers heterogeneous network conditions and the FedAdagrad, FedYogi, and FedAdam algorithms that provide adaptive federated optimization [15]. FLScalize is implemented based on the Flower framework that supports the connection function between the FL client and server.

A. FL PLATFORMS AND FRAMEWORKS

Various platforms and frameworks have been developed for studying FL [16]. TensorFlow Federated (TFF) is an FL framework that provides only basic FL functions based on TensorFlow [17]; however, a separate system must be built to apply custom data and models to FL tasks. FATE uses Kubernetes and Docker to deploy an FL environment [4]; however, its complex system design makes it difficult to use in practice, and only FL libraries belonging to the platform can be used. FedScale [3] and EasyFL [6] can easily simulate FL with a few code lines; however, functions for continuous integration, deployment, and training are not supported and managed. Therefore, existing platforms and frameworks suffer from some disadvantages.

FLScalize aims to overcome these advantages by implementing FL client and server based on the Flower framework. Flower has better FL client and server scheduling capabilities than those of other FL platforms, and it can perform FL rounds by simply connecting the server and multiple clients [2]. However, it only supports an FL communication function and does not support the lifecycle function to manage FL operations. In this light, FLScalize extends the Flower framework function to enable the easy application of custom data and models to an FL environment, unlike in the case of other FL platforms, as shown in **Table 2**. Further, it enables FL simulations by configuring various FL task environments that can occur in the real world. It supports a manager component that can continuously manage and track the FL client and server and provide FL lifecycle management that enables continuous integration (CI), continuous deployment (CD), and continuous training (CT) functions for the latest version of the data and models.

B. KUBERNETES

MLOps aims to serve models accurately and reliably by creating a pipeline to deliver ML models [18]. To create a pipeline, CI, CD, and CT functions are supported. FLScalize creates CI, CD, and CT pipelines in the Kubernetes environment to support FL environments that can occur in the real world.

1) FEATURES OF KUBERNETES

Containerizing and running various applications can be complex as many containers are deployed across multiple servers (nodes). Kubernetes can solve these problems by providing a method for scheduling and deploying containers, thereby enabling scaling by managing the status and lifecycle of the entire cluster [19]. Implementing container-based applications using Kubernetes affords the following advantages [20], [21], [22]:

- Scalable: Kubernetes enables defining and deploying complex containerized apps on clusters of multiple servers (nodes). Additionally, container health can be automatically monitored and maintained while scaling the application.
- Portable: Because container apps are isolated on multiple nodes, they can remain portable when using Kubernetes. It is possible to move from a local machine to a production environment while maintaining consistency across on-premises, hybrid, and multiple cloud environments.
- Extensible: Developers and enterprises from a huge open source community can actively add extensions and

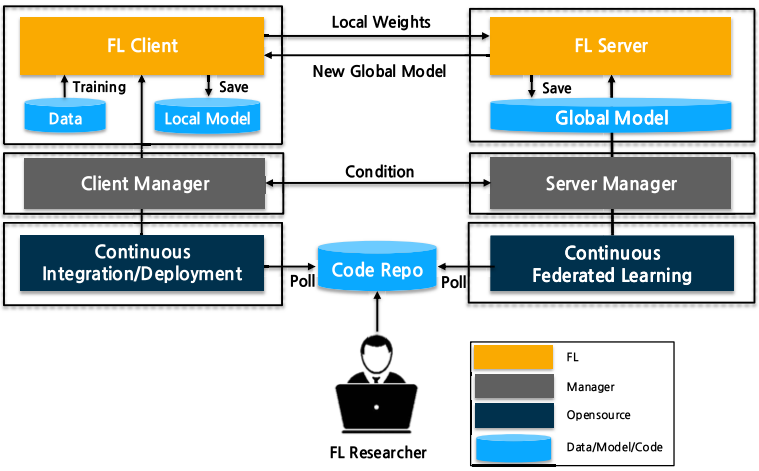


FIGURE 2. Design and architecture of FLScalize including expanded components necessary for managing the FL lifecycle for stable FL task operation.

plug-ins to add security, monitoring, management, and other features.

FLScalize provides an FL environment that can occur in the real world in a Kubernetes cluster environment consisting of multiple nodes to create multiple scalable container apps and to support efficient and portable operation management, and it provides an automated FL operating environment in which CI, CD, and CT functions are added to enable extensible FL lifecycle management.

2) CI/CD

The CI/CD pipeline automates the following steps: (1) build, (2) test, (3) release, (4) deployment, and (5) validation and compliance [23]. Automation tools that support CI and CD include Spinnaker [24], Jenkins X [25], Tekton [26], and Argo CD [27]. ArgoCD supports GitOps-style deployment. When desired settings are changed and pushed to Git, the status of the Kubernetes cluster is automatically synchronized with that defined in Git [27]. In other words, the application is automatically deployed in the desired status to the designated target environment. Because it enables code version management, previously deployed codes can be reused. FLScalize connects to ArgoCD-based Git to provide CI and CD operations.

3) CT

The CT automates and repeats the process of training new data and deploying new models. It manages the performance of the model by providing a continuous training environment [28]. Airflow is an open source application that sequentially guarantees the execution order of tasks and supports the creation, scheduling, and monitoring of workflows [29]. FLScalize provides the CT function that can continuously perform FL tasks using Airflow.

To create the FL lifecycle management platform, the CI, CD, and CT pipeline in the FL environment is required. FLScalize provides CI, CD, and CT functions in the Kubernetes environment by using open source applications.

III. FLSCALIZE SYSTEM DESIGN AND ARCHITECTURE

FLScalize is configured to easily apply custom data and models to FL environments that can occur in the real world. It also provides a manager component that manages the FL client and server and can perform CI, CD, and CT functions centered on the latest code connected to Git. FLScalize can perform system and data heterogeneity simulations with this function. **Fig. 2** shows the composition of all components of FLScalize.

A. FL CLIENT AND FL SERVER

The FL client participating in the FL round and the FL server that creates the FL round task are implemented based on the Flower framework, and they communicate with each other through gRPC [2]. In the FL client, as shown in **Fig. 3**, data_load and model_build functions can be created to enable AI model researchers to easily apply their custom data and models to the FL environment. To apply custom data, the client_data module was configured. It supports the function of loading client data and adds a data partition function to hold different data for each client ID. In addition, data heterogeneity can be simulated by configuring IID and non-IID cases. When applying a custom model, the model structure, optimizer, and loss function to be used in the client_model module can be set. Finally, as shown in **Fig. 4**, custom FL can be performed by creating a model using the custom data added in **Fig. 3** in the app module of the FL client.

The FL server creates FL rounds by setting the initial global model, hyperparameters, and aggregation algorithm. For each round, the FL client sends the weights of the local model to the FL server, and the FL server creates a global model by aggregating several local weights and sends it back to the FL client. By repeating this process as many times as the set number of rounds, the FL client performs local training with the local model and the FL server creates the global model. The local model, which is the model that is trained on the data from each individual client, is stored and managed on each client. The global model is stored and managed according

to the implementation of FL on the server. This means that the server is responsible for managing the global model and ensuring that it is up to date with the latest local models from each client.

```
# Load the dataset partitions
def data_load(all_client_num, fl_client_num, dataset, skewed, skewed_spec, balanced):

 if dataset == 'cifar10':
   (X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
                                                                    client_data module
 # IID/Non-IID Partition
 (X_train, y_train), (X_test, y_test) = data_partition(X_train, y_train, X_test, y_test, skewed, \
                                skewed_spec, balanced, FL_client_num, all_client_num, dataset)

 return (X_train, y_train), (X_test, y_test)

# Create Custom Model
def model_build():
  model_VGG16 = tf.keras.applications.VGG16(weights = 'imagenet', include_top = False,input_shape = (32,32,3))

  for layer in model_VGG16.layers:
    layer.trainable = False
                                                                    client_model module
  x = tf.keras.layers.Flatten()(model_VGG16.output)
  x = tf.keras.layers.Dense(4096, activation='relu')(x)
  x = tf.keras.layers.Dense(1000, activation='relu')(x)
  predictions = tf.keras.layers.Dense(10, activation = 'softmax')(x)


  model = tf.keras.Model(inputs = model_VGG16.input, outputs = predictions)
  model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])

  return model
```

FIGURE 3. Example of easily creating custom data and custom model in client_data module.

```
# Load Client Custom Data
async def flower_client_start():
  logging.info('FL learning ready')
  global status

  # data setting parameter
  all_client_num = 5 # total client number
  dataset = 'fashion_mnist' # dataset
  skewed = False # data partition1: Each client has only one class (or two/three classes)
  skewed_spec = 'skewed_one'
  balanced = False # data partition2: Each client is randomly distributed in different sizes

  # Client Data
  (x_train, y_train), (x_test, y_test) = client_data.data_load(all_client_num, status.FL_client_num, dataset, skewed, skewed_spec, balanced)

# Load Client Custom Local Model
def build_model(dataset):

  if dataset == 'cifar10':
    model = client_model.model_cnn()

  elif dataset == 'mnist':
    model = client_model.model_ResNet50()

  elif dataset == 'fashion_mnist':
    model = client_model.model_VGG16()

  return model
```

FIGURE 4. Example of applying custom data and models in the FL client's app module.

B. CLIENT MANAGER AND SERVER MANAGER

As shown in **Fig 5**, the client manager and server manager play the role of managing FL client and server, respectively. The client manager continuously checks the online and training status of the FL client and the online status of the FL server. To reliably check the status of each component, the client manager continuously checks the status of the FL client and server through asynchronous API communication. The server manager checks whether the FL server has started the FL round and transfers the status information of the FL server to the client manager. Further, it manages the version of the global model created for each FL round. The client manager receiving the information from the server manager transfers a trigger to participate in the FL round to the FL client and manages the FL round participation.

C. CI/CD

ArgoCD, which provides GitOps CI/CD functions, is used to deploy the latest versions of the FL client, client manager, and server manager. As shown in **Fig. 6**, the client pod that group the FL client and client manager containers, and server manager pod are created by connecting the Git repo of each

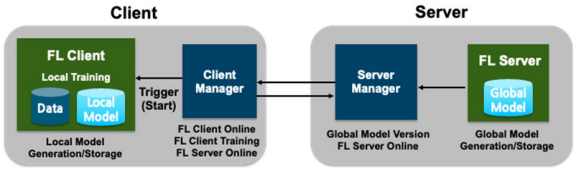


FIGURE 5. Communication between the FL client manager and the FL server manager.

component with ArgoCD. Two categories of FL simulation environments are provided by configuring the FL client environment that can occur in the real world. First, a system heterogeneity simulation environment can be implemented by setting resources such as the CPU, GPU, and Memory in the client pod. In addition, the number of clients can be easily set by duplicating clients, and an ID is given to each client so that each client can hold different data. In other words, the simulation environment supports data heterogeneity. In this way, an FL environment that can occur in the real world can be configured for various FL tasks.

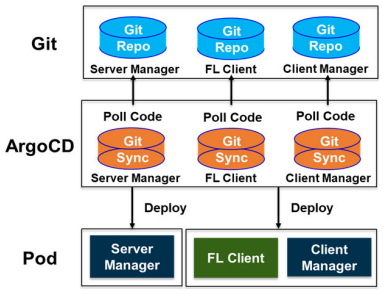


FIGURE 6. Process that supports CI/CD using ArgoCD linked with Git.

D. CONTINUOUS FEDERATED LEARNING

Airflow can create multiple tasks by supporting workflow functions that support scheduling and monitoring. In addition, it recognizes the latest version of code that is continuously updated in conjunction with Git. FLScalize uses Airflow to create various FL tasks, continuously deploy the latest version of the FL server pod, and execute FL rounds. As shown in **Fig. 7**, client pods in the online status receive start information from FL server pods and participate in FL rounds, thereby enabling continuous federated learning (CFL) to be performed.

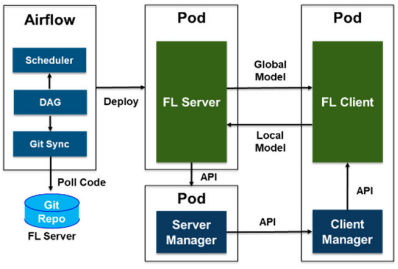


FIGURE 7. Process that connects the FL server Git repository using Airflow to continuously update and deploy the latest FL server code, enabling CFL.

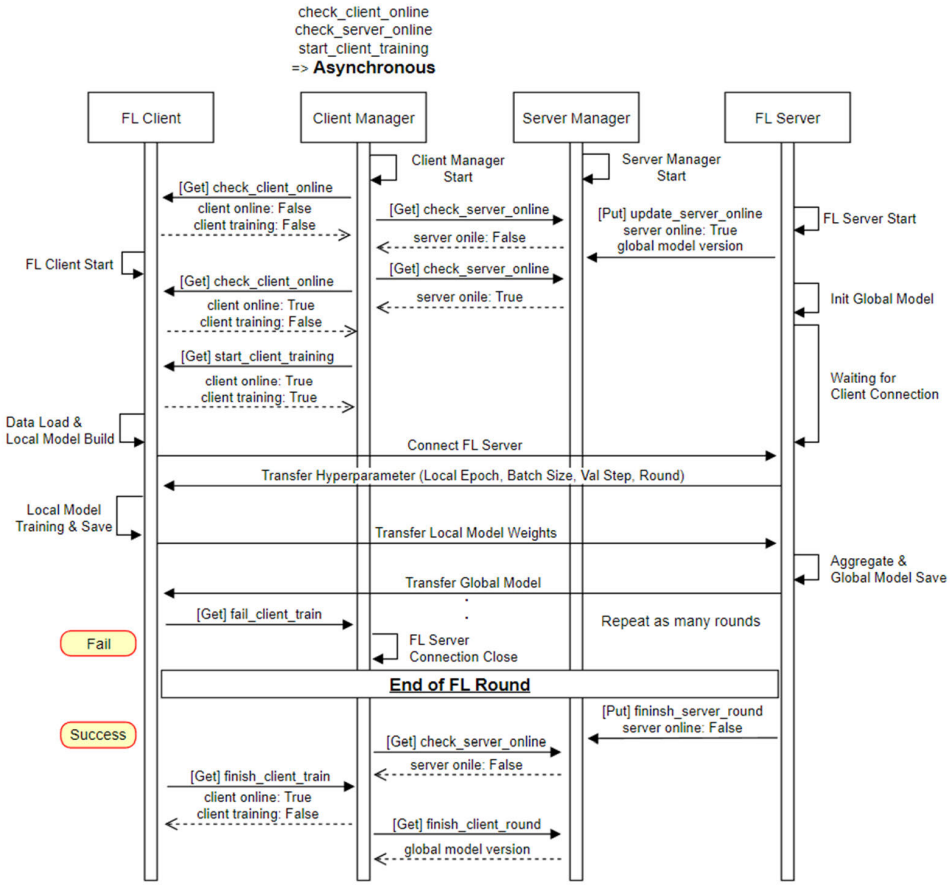


FIGURE 8. FL lifecycle process of FLScalize for component communication configuration, API and status flow.

E. FLSCALIZE LIFECYCLE MANAGEMENT COMMUNICATION PROCESS

FL client, client manager and server manager are connected to each git repo and continuously integrated and deployed through ArgoCD. In addition, the FL server periodically creates rounds of FL tasks through airflow to perform continuous federated learning. When each component is distributed like this, each component of FLScalize communicates as **Fig. 8**. **Table 3** describes the API functions of FLScalize.

1) The client manager continuously checks the online status of the FL client and server through asynchronous API communication. Asynchronous API communication was used to solve the problem of inconsistency between the status information of the FL client and server.
2) If the FL client and server are started, the online status is returned as True, and the FL server transmits the global model version to the server manager. However, if the FL server is not online, FL cannot be executed.
3) The FL server also creates an init global model and waits until all the clients that will participate in the FL round are connected.
4) When the client and server online status is True and the client training status is False, the client manager sends a trigger to the FL client to participate in the FL round.
5) The FL client creates a local model and loads the data to be trained. The hyperparameter information of the local model is received from the FL server, and local training is performed based on it.
6) The FL client performs local training for each round, saves the local model, and delivers the updated local weights to the FL server. The FL server creates and stores a global model by aggregating the local weights of clients participating in FL rounds. This is repeated for the set number of rounds.
7) If an FL client encounters an error during a round (power, Internet, etc.), the FL client terminates the connection with the FL server. If specific clients lose communication with the server for some reason, FL is performed with the rest of the clients excluding the disconnected clients. If the minimum number of participating clients required for the FL round set by the server is not met, the server continues to wait. However, the FL is forcibly terminated according to the set timeout.
8) When the FL round is successfully completed, the FL server transmits the termination API to the server manager, and the FL client notifies the client manager that local training has been completed. The client manager receives the successfully completed global model

version information from the server manager that manages the global model version.

9) The next FL task proceeds in the same way as above when only the FL server is started. As long as the FL client is not terminated, it automatically participates in FL rounds by the client manager and can manage the FL lifecycle.

TABLE 3. API function definition of each component of FLScalize.

Request Component	API Name	Description
Client Manager	check_client_online	Check online status of FL Client
Client Manager	check_server_online	Check online status of FL Server
Client Manager	start_client_training	Trigger FL Client's FL round participation
Client Manager	finish_client_round	Transfer FL Client's FL end to Server Manager
FL Client	fail_client_train	Error while participating in FL round
FL Client	finish_client_train	Complete successfully FL round
FL Server	update_server_online	Transfer online information of FL Server
FL Server	finish_server_round	Transfer offline information of FL Server

TABLE 4. Specifications of servers used in experiment.

Server Name	CPU	GPU
D Server	Intel i9-9900KF (8core)	NVIDIA®2080Ti (2)
E Server	Intel Xeon Silver 4214R (12core)	NVIDIA®3090Ti (2)
X Server	Intel i9-10980XE (18core)	NVIDIA®3070Ti (2)

As such, FLScalize additionally configures the client and server manager components to continuously check the status of the FL client and server and to support the CFL function. In addition, because local models and global model that are created for each FL round can be managed, it is possible to check and track which model version has better performance. Furthermore, lifecycle management for the entire FL process is possible.

IV. EXPERIMENTAL ENVIRONMENT SETUP

FLScalize offers an FL lifecycle environment that effectively manages the status of both FL clients and servers, while enabling seamless CI/CD/CFL workflows for real-world FL tasks. FLScalize also performs several FL tasks based on various open datasets and models. Our platform manages the FL lifecycle by assigning system heterogeneity and data heterogeneity to these FL tasks. Various FL lifecycle management experiments were conducted in the Kubernetes environment, as described below.

A. KUBERNETES CLUSTER SERVER SPECIFICATIONS

All FL task simulations were conducted with three servers (nodes) in one Kubernetes cluster. The three nodes have different specifications, as listed **Table 5**.

TABLE 5. Dataset and model used in FLScalize experiment. Each row lists one FL task.

Dataset	Model	Sample
CIFAR-10	CNN (2Conv+2FC)	60,000
MNIST	ResNet50	70,000
FASHION MNIST	VGG16	70,000

B. DATASET/MODEL

The data and model shown in **Table 5** were used for the system and data heterogeneity simulations of FLScalize, and they were set as three FL tasks. To perform data heterogeneity simulation, IID and non-IID partitions were divided into client data. IID consisted of balanced data that was distributed to each client as a dataset of the same size, and non-IID consisted of imbalanced data, skewed data, and imbalanced/skewed combined data. The non-IID data composition is described in **Table 6**.

TABLE 6. Partition configuration of Non-IID dataset.

Non-IID	Description
Imbalanced Data	Each client is randomly distributed in different sizes
Skewed Data	Each client has only one class (or two/three classes)
Imbalanced /Skewed Data	Each client has only one class and is distributed in different sizes

The data partition configuration is required to implement data heterogeneity. D_l denotes the size of the total data; C_A, the total number of clients; and C_i, the data index list of each client. A balanced data partition holds different data for each client and has the same data size, as shown in (1).

$$BC_i = \{x | x \in N, [\frac{D_l}{C_A} \times i, \frac{D_l}{C_A} \times (i+1)]\} \quad (1)$$

An imbalanced data partition holds different data for each client, and the size of the data held by each client is set

randomly, as shown in (2).

$$\mathrm{IBC_i} = \{\mathrm{x}|\mathrm{x} \in \mathrm{N}, [rand(\frac{D_l}{C_A} \times i), rand(\frac{D_l}{C_A} \times (i+1))]\} \tag{2}$$

A skewed data partition has different labels for each client, as shown in (3). $\mathrm{L_i}$ denotes the label list of each client; $\mathrm{D_L}$, the label of all data; and DS, the list of specific labels.

$$\begin{aligned} \mathrm{DS} &= \{D_L \cap L_i\} \\ \mathrm{SC_i} &= \{\mathrm{x}|\mathrm{x} \in \mathrm{DS}, [\frac{D_l}{C_A} \times i, \frac{D_l}{C_A} \times (i+1)]\} \end{aligned} \tag{3}$$

FLScalize can deal with data issues that can occur in real-world FL environments by providing IID and non-IID data partition functions for various FL tasks.

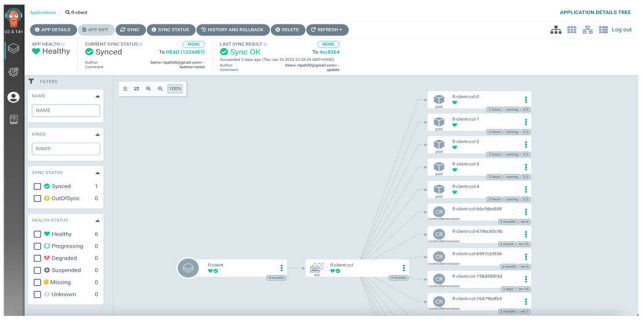


FIGURE 9. Deploying and running five client pods using ArgoCD.

C. AGGREGATION ALGORITHM AND HYPERPARAMETER

FLScalize uses the FedAvg algorithm to aggregate the local weights of multiple clients. In addition, in all experiments, the FL hyperparameters were set as follows: 10 for the local epoch, 32 for batch size, 32 for the evaluation step for each client evaluation, and 20 for the round. In addition, ADAM was used as the optimizer of the local model. The loss function used in the model is categorical crossentropy.

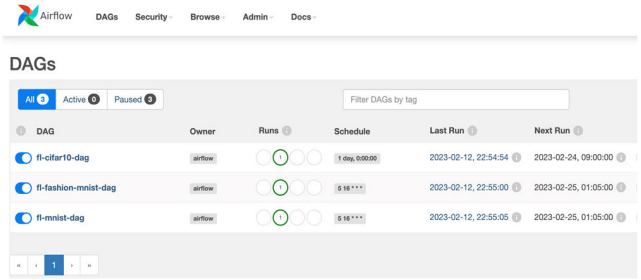


FIGURE 10. Creating three FL tasks (CIFAR-10, MNIST, and FASHION MNIST) to support a workflow that creates an FL round.

V. IMPLEMENTAION AND RESULT

To test FLScalize, the data and models listed in **Table 5** were designated as FL tasks; the manager component was applied; and CI, CD, and CFL were performed in the system and data heterogeneity simulation environment. This experiment was performed to confirm that operation management for various FL tasks was possible and to introduce an FL lifecycle management platform.

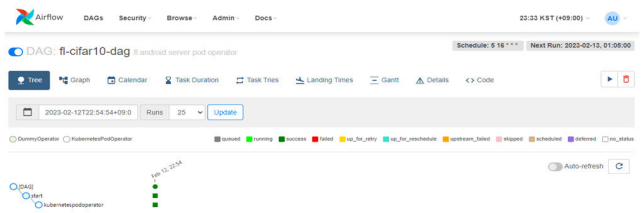


FIGURE 11. By using Airflow, an FL task for CIFAR-10 is deployed to the FL server that creates FL round.

A. CLIENT AND SERVER MANAGER POD DEPLOYMENT

The client pod includes the FL client and client manager containers. As shown in **Fig. 9**, ArgoCD was used to deploy client pods and to easily set the number of client pods. In this experimental environment, the number of client pods for each FL task was set to 5. As resources such as the CPU, GPU, and Memory can be specified for each client pod, a heterogenous system environment can be implemented. In addition, by assigning a unique ID to each client pod, each client can have a different dataset, and data heterogeneity can be simulated using data partitions. The server manager pod checks the status of the FL server and delivers the online status of the FL server to the client manager; it was also deployed using ArgoCD.

B. FL SERVER POD CI/CD AND CFL

FLScalize uses Airflow to periodically deploy and execute FL server pods to support a workflow that creates various FL tasks, as shown in **Fig. 10**. The deployed FL server pod is executed to create FL rounds. The client pod recognizes the start status of the FL server pod from the client manager, and it receives a trigger to participate in FL round. This FL task is performed as many times as the number of rounds set on the FL server. As shown in **Fig. 11**, periodic and automated CFL can be performed by setting the time for creating an FL task.

TABLE 7. Training time according to system specifications of each client group. Each round time and local training time represent average values.

Client Group	System	Each Round Time	Local Training Time	Total Time
Group 1	CPU 1, Memory 2 GB	66.7 s	48.2 s	1,346.4 s
Group 2	CPU 3, Memory 4 GB	60.5 s	44.3 s	1,224.1 s
Group 3	CPU 1, GPU 1, Memory 2 GB	61.1 s	44.7 s	1,243.4 s

C. HETEROGENEOUS SIMULATION

1) SYSTEM HETEROGENEITY

To compare the resource performance of FL clients that can occur in the real world, the clients were divided into three

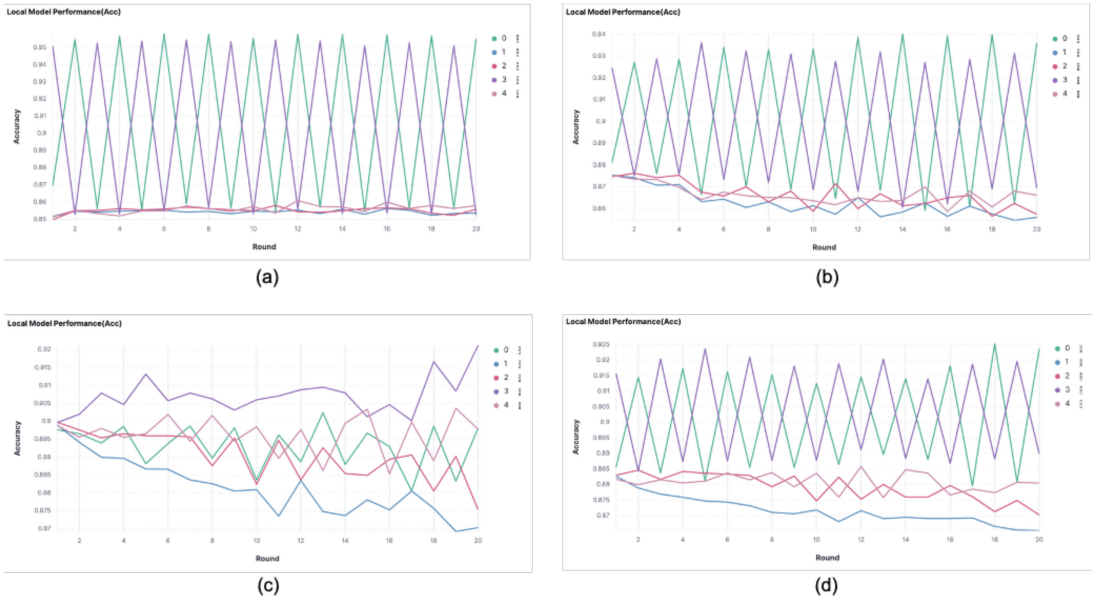


FIGURE 12. Local model accuracy performance of five clients on non-IID data partition of CIFAR-10 dataset: (a) skewed one class, (b) skewed two class, (c) skewed three class, and (d) imbalanced/skewed three class.

TABLE 8. Accuracy comparison of local and global model according to data heterogeneity. Local model averages performance of each client, and performance of global model includes all labels.

Data Heterogeneity	CIFAR-10		FASHION MNIST		MNIST	
	Local Model	Global Model	Local Model	Global Model	Local Model	Global Model
IID	91.8%	91.4%	96.6%	89.8%	95.6%	91.7%
Imbalanced	91.3%	91.1%	84.2%	80.6%	94.9%	91.5%
Skewed One Class	84.2%	81.9%	14.7%	32.1%	26.8%	13.4%
Skewed Two Class	88.6%	88.3%	25.2%	12.7%	25.9%	25.2%
Skewed Three Class	90.4%	89.7%	59.3%	55.9%	38.9%	33.6%
Imbalanced/ Skewed Three Class	89.3%	88.5%	49.1%	47.3%	29.9%	25.7%

client groups that each consisted of five client pods but had different resources. Based on client groups with different resources, a system heterogeneity simulation was performed using the CIFAR-10 dataset and CNN model, and the time for FL execution was evaluated as shown in **Table 7**. Each round time is the time for aggregating the weights of clients for each FL round in the FL server, and the average of all rounds is taken. The local training time is the average of the local training value of all clients. The total time is the time for which FL has been performed for the set round. As shown in **Table 7**, Group 2 showed a difference of 6 s for each round time, 4 s for local training time, and 122 s for total time compared to those of Group 1 owing to the difference in system performance. Although the GPU was allocated to Group 3 but not to Group 2, the time cost of Group 2 was higher as its model had fewer parameters.

As such, it is possible to check which data and model are suitable according to the resource performance of the client by configuring the FL client resource environment that can occur in the real world using FLScalize.

2) DATA HETEROGENEITY

To perform the data heterogeneity simulation, IID and non-IID data partitions were constructed for the dataset in **Table 5**. In addition, this FL task was created using a model corresponding to the dataset, and the accuracy was compared according to the data heterogeneity for each FL task. The average of each client's local model was calculated for evaluating the performance of the local model. To evaluate the performance of the global model, a new untrained dataset was used, and the dataset containing all labels was constructed for evaluation. As shown in **Table 8**, for the non-IID case with the data partition, the performance of the local and global models was lower than those in the IID case. Furthermore, even in the non-IID case, the performance deteriorated in the presence of more severe data heterogeneity. As shown

in **Fig. 12**, each client performs local training with different classes and data sizes in a non-IID case, and the accuracy is different for each local model of the client. In particular, in the situation where one class is biased, the accuracy of a specific client did not converge, and the oscillation width was larger than that in the other skewed situation. In the case of severe data heterogeneity, training did not work properly. When the skewed data partition case was assigned with the Fashion MNIST and MNIST datasets, compared to the IID case, the performance degradation was more severe than that with the CIFAR-10 dataset. Even with the same data heterogeneity, the performance could vary greatly depending on the data and model. FLScalize supports the data partition function to handle data issues that may occur in the real world for each FL task. FL researchers who intend to use FLScalize can perform the FL task they want by applying suitable data heterogeneity.

FLScalize can simulate system heterogeneity by assigning and deploying different system performances for each client, and it can check the time cost according to the client's system performance. In addition, by providing IID and non-IID data partition environments, it enables data heterogeneity simulations according to the dataset and evaluation of their performance.

We incorporated the features highlighted by FLScalize in contributions 1 to 3 in the system and data heterogeneity simulation, and evaluated FL performance results with varying system and data configurations across clients. To effectively manage the results of these FL tasks, we developed an FL lifecycle management platform capable of tracking and managing various FL task outcomes. An FL task environment that can occur in the real world can be configured using FLScalize, and custom data and models can be easily applied to perform FL lifecycle management with CI, CD, and CFL functions that support simulations with system and data heterogeneity.

FLScalize simulated FL system heterogeneity by configuring a cloud with servers that we own. However, due to the limited computing resources available, there may be limitations in the extent to which our cloud environment can accurately reflect the performance of FL in other cloud environments.

VI. CONCLUSION AND FUTURE WORK

Various FL platforms and frameworks have emerged with the increasing focus on data privacy. However, various industries and services lack a platform that can support FL operation management. To introduce a platform that supports FL operation, an FL environment that can occur in the real-world was constructed. The proposed FLScalize can easily apply custom data and models in a centralized environment to an FL environment and create various FL tasks. We provide an FL lifecycle management environment that enables continuous status management, integration, deployment, and training. We support a manager component that continuously checks and manages the status of FL the client and server, a multiclient resource configuration environment, and an environment that creates various FL tasks and performs FL rounds periodically. FLScalize can be used to simulate system and data heterogeneity as it includes these functions, and it can manage the FL lifecycle for custom data and models for FL tasks.

FLScalize offers several practical benefits in the implementation of FL. Firstly, it provides a structured framework for managing the entire FL process, including model training, deployment, and updating. This can help improve the efficiency and effectiveness of FL by streamlining the workflow and reducing the chances of errors or oversights. Secondly, FLScalize enables better tracking and management of FL task results, including performance metrics, data usage, and model updates. This allows stakeholders to gain insights into the performance of FL models and make informed decisions about the allocation of computing resources and data. Finally, FLScalize can improve the reliability and reliability of FL systems by managing models and data throughout the FL process.

FLScalize has performed FL lifecycle management in a cloud environment by simulating system and data heterogeneity. However, the study has a limitation in terms of performing FL lifecycle management in a real device environment, which requires additional supplements. Thus, further research is needed to overcome this limitation and make FLScalize more versatile in managing FL tasks in real-world FL projects.

In future work, the FL lifecycle management platform can be extended to support a real-world multiclient and server decoupled environment, which will require additional research on how to efficiently manage and monitor multiple FL clients with diverse computing capabilities and security requirements. This can involve exploring techniques for distributed training, such as federated averaging, and designing secure communication protocols to protect sensitive data exchanged between clients and the server. Additionally, the FL operation platform can be enhanced to provide more advanced services, such as automated hyperparameter tuning, model compression, and continual learning, to improve the overall performance and efficiency of the FL process. Finally, integrating the platform with existing cloud computing and edge computing infrastructures can provide more scalability and flexibility for managing and deploying FL models in real-world scenarios.

REFERENCES

[1] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, "Federated learning for the Internet of Things: Applications, challenges, and opportunities," *IEEE Internet Things Mag.*, vol. 5, no. 1, pp. 24–29, Mar. 2022.

[2] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, and P. P. B. de Gusmão, "Flower: A friendly federated learning framework," 2022, *arXiv: 2103.08942*.

[3] F. Lai, Y. Dai, S. Singapuram, J. Liu, X. Zhu, H. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking model and system performance of federated learning at scale," in *Proc. Mach. Learn. Res.*, vol. 162, 2022, pp. 11814–11827.

[4] *FATE*. Accessed: Jan. 9, 2023. [Online]. Available: https://fate.fedai.org

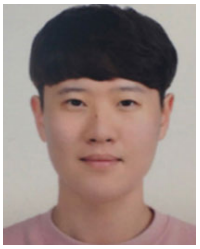
[5] J. Xie, K. Zhang, and A. T. Frank, "PyShifts: A PyMOL plugin for chemical shift-based analysis of biomolecular ensembles," *J. Chem. Inf. Model.*, vol. 60, no. 3, pp. 1073–1078, Mar. 2020.

[6] W. Zhuang, X. Gan, Y. Wen, and S. Zhang, "EasyFL: A low-code federated learning platform for dummies," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13740–13754, Aug. 2022.
[7] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervent.*, 2018, pp. 92–104.
[8] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang, "Federated learning-powered visual object detection for safety monitoring," *AI Mag.*, vol. 42, no. 2, pp. 19–27, Oct. 2021.
[9] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6532–6542, Oct. 2020.
[10] Y. Tian, Y. Wan, L. Lyu, D. Yao, H. Jin, and L. Sun, "FedBERT: When federated learning meets pre-training," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 4, pp. 1–26, Aug. 2022.
[11] S. Makinen, H. Skogstrom, E. Laaksonen, and T. Mikkonen, "Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?" in *Proc. IEEE/ACM 1st Workshop AI Eng. Softw. Eng. AI (WAIN)*, May 2021, pp. 109–112.
[12] Y. Zhao, A. S. Z. Belloum, G. M. Da Costa, and Z. Zhao, "MLOps scaling machine learning lifecycle in an industrial setting," *Int. J. Ind. Manuf. Eng.*, vol. 16, no. 5, pp. 143–153, 2022.
[13] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3347–3366, Apr. 2023.
[14] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.
[15] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *Proc. ICLR*, 2021, pp. 1–38.
[16] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140699–140725, 2020.
[17] I. Kholod, E. Yanaki, D. Fomichev, E. Shalugin, E. Novikova, E. Filippov, and M. Nordlund, "Open-source federated learning frameworks for IoT: A comparative review and analysis," *Sensors*, vol. 21, no. 1, p. 167, Dec. 2020.
[18] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying DevOps practices of continuous automation for machine learning," *Information*, vol. 11, no. 7, p. 363, Jul. 2020.
[19] Kubernetes Community. (2022). *Kubernetes: Production-Grade Container Orchestration*. [Online]. Available: https://kubernetes.io/
[20] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for kubernetes edge clusters," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 958–972, Mar. 2021.
[21] J. J. Barriga, J. Sulca, J. León, A. Ulloa, D. Portero, J. García, and S. G. Yoo, "A smart parking solution architecture based on LoRaWAN and Kubernetes," *Appl. Sci.*, vol. 10, no. 13, p. 4674, Jul. 2020.
[22] D. Jorge-Martinez, S. A. Butt, E. M. Onyema, C. Chakraborty, Q. Shaheen, E. De-La-Hoz-Franco, and P. Ariza-Colpas, "Artificial intelligence-based Kubernetes container for scheduling nodes of energy composition," *Int. J. Syst. Assurance Eng. Manage.*, vol. 2021, pp. 1–9, Jul. 2021.
[23] S. A. I. B. S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, May 2018, pp. 156–161.
[24] Spinnaker. (2022). *Spinnaker: Continuous Delivery Platform for Enterprise*. [Online]. Available: https://spinnaker.io/
[25] X. Jenkins. (2022). *Jenkins X: Continuous Delivery for Kubernetes*. [Online]. Available: https://jenkins-x.io/
[26] Tekton. (2022). *Tekton: The Kubernetes-Native Continuous Integration and Delivery Solution*. [Online]. Available: https://tekton.dev/
[27] Ramadoni, E. Utami, and H. A. Fatta, "Analysis on the use of declarative and pull-based deployment models on GitOps using Argo CD," in *Proc. 4th Int. Conf. Inf. Commun. Technol. (ICOIACT)*, Aug. 2021, pp. 186–191.
[28] M. M. John, H. H. Olsson, and J. Bosch, "Towards MLOps: A framework and maturity model," in *Proc. 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2021, pp. 1–8.
[29] *Apache Airflow*. Accessed: Jan. 14, 2023. [Online]. Available: https://airflow.apache.org/

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 Gachon University, Seoul, South Korea.

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

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 University. His current research interests include federated learning, reinforcement learning, generative model, automation, multimodal analysis, and digital twin.



JINSOO KIM received the B.S. degree in computer engineering and the M.S. degree in IT convergence engineering from Gachon University, Seoul, South Korea, in 2018 and 2020, respectively.

From 2018 to 2022, he was a Researcher with the Gachon Cognitive Computing Laboratory. He joined TVSTORM as a Development Engineer to research healthcare signage solutions. His research interests include federated learning and meta-learning.

Mr. Kim received an award from the 4th University App Development Challenge (K-Hackathon). His project received the Excellent Project Award from the Gyeonggi Big Data Challenge.

KWANGKEE LEE received the B.S., M.S., and Ph.D. degrees in electronics engineering from Yonsei University, Seoul, South Korea, in 1986, 1988, and 1993, respectively.

From 1994 to 2014, he was a Technical Researcher with the Samsung Advanced Institute of Technology, and he was promoted to Group Leader of the SW Platform and Next-Generation Strategy Team. From 2016 to 2019, he was an Industrial Convergence PD for R&BD planning with the Ministry of Industry, Industrial Technology Evaluation and Management Institute. From 2019 to 2022, he was the Director of the Research and Development Team, Tivistom Company Ltd. He is currently a Researcher with Innopia Technologies Inc. He won a proud Samsung Prize, in 2006. He received the GSMA EMWC (M2M) Best Embedded Mobile End-to-End Service Winner, in 2010.

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 Ubiquitous Computing and the Software Solutions Laboratory. 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, IT College, Gachon University. He has been the Director of the Gachon Artificial Intelligence Technology Center, since 2016. His research interests include cognitive computing, healthcare advisor, the IoT platform, and industry transformation.