future internet

MDPI

Article

Multi-Level Split Federated Learning for Large-Scale AIoT System Based on Smart Cities

Hanyue Xu [1,2], Kah Phooi Seng [1,3,4,*], Jeremy Smith [2] and Li Minn Ang [4]

[1] School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University, Suzhou 215000, China; hanyue.xu19@student.xjtlu.edu.cn
[2] Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, UK; j.s.smith@liverpool.ac.uk
[3] School of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[4] School of Science, Technology and Engineering, University of the Sunshine Coast, Petrie, QLD 4502, Australia; lang@usc.edu.au
* Correspondence: jasmine.seng@xjtlu.edu.cn

Abstract: In the context of smart cities, the integration of artificial intelligence (AI) and the Internet of Things (IoT) has led to the proliferation of AIoT systems, which handle vast amounts of data to enhance urban infrastructure and services. However, the collaborative training of deep learning models within these systems encounters significant challenges, chiefly due to data privacy concerns and dealing with communication latency from large-scale IoT devices. To address these issues, multi-level split federated learning (multi-level SFL) has been proposed, merging the benefits of split learning (SL) and federated learning (FL). This framework introduces a novel multi-level aggregation architecture that reduces communication delays, enhances scalability, and addresses system and statistical heterogeneity inherent in large AIoT systems with non-IID data distributions. The architecture leverages the Message Queuing Telemetry Transport (MQTT) protocol to cluster IoT devices geographically and employs edge and fog computing layers for initial model parameter aggregation. Simulation experiments validate that the multi-level SFL outperforms traditional SFL by improving model accuracy and convergence speed in large-scale, non-IID environments. This paper delineates the proposed architecture, its workflow, and its advantages in enhancing the robustness and scalability of AIoT systems in smart cities while preserving data privacy.

Keywords: federated learning; split learning; split federated learning; artificial intelligent internet of things; edge computing

Citation: Xu, H.; Seng, K.P.; Smith, J.; Ang, L.M. Multi-Level Split Federated Learning for Large-Scale AIoT System Based on Smart Cities. *Future Internet* **2024**, *16*, 82. https://doi.org/10.3390/fi16030082

Academic Editors: Qiang Duan and Zhihui Lu

Received: 30 January 2024
Revised: 21 February 2024
Accepted: 26 February 2024
Published: 28 February 2024

1. Introduction

With the acceleration of urbanization, smart cities are proposed to utilize various artificial intelligence (AI) technologies or urban infrastructure to integrate artificial intelligence Internet of Things (AIoT) systems, improve resource utilization efficiency, optimize city management and services, and achieve the idea of the Internet of Everything. By analyzing and processing the massive historical and real-time data generated by IoT devices such as sensors, AI technology can make more accurate predictions about future devices and user habits, such as smart grid [1], smart transportation [2], and smart healthcare [3]. On the other hand, hyperscale data connected through IoT can also lay the foundation for deep learning in AI. However, with the continuous development of big data technology, the meaning of the data generated in smart cities for everyone is no longer insignificant information but a digital asset. For example, the user driving habits data of smart vehicles need user authorization to be used for model training and learning of AI technology [4]. Therefore, the need to train deep learning models without aggregating and accessing sensitive raw data on the client side is a major challenge that AIoT systems need to solve for multi-client collaborative learning.

In recent years, the concept of distributed collaboration machine learning (DCML) has been proposed to solve the above challenges, including federated learning [5–7] and split learning [8–10]. Different from traditional centralized machine learning, DCML addresses data privacy challenges by collaborating with multiple IoT devices (clients) to train machine learning or deep learning models in collaboration with a central server, without sharing local data generated by individual IoT devices. In federated learning (FL), the global model is constructed by aggregating the model parameters trained by the local model on each client through the cloud server. In split learning (SL), the deep learning model is split into two parts: the first few layers are trained by the IoT device (client), and the bottom layer is calculated by the central server (cloud), which is mainly used to solve the problem of the limited computing resources of IoT devices. However, the performance of split learning decreases as the number of clients increases, and federated learning is not suitable for IoT devices with limited resources, so they all have limitations in large-scale AIoT systems. Therefore, split federated learning [11–13] is proposed to combine the advantages of SL and FL to make an AIoT system parallel in data and model training, which not only considers the problem of limited client resources but also reduces the influence of the number of clients on the performance of the model. Furthermore, it does not lose the protection of data privacy and the robustness of the model.

Although split federated learning has become a new paradigm for future collaborative learning in AIoT systems, the performance and efficiency of model training are greatly reduced due to the large number of clients contained in large AIoT systems, the different transmission distances of different clients, and the insufficient stability of single cloud center server nodes. During the training process, there may be statistical heterogeneity and system heterogeneity among AIoT devices that have different computing and storage resources, and the generated data are not independent and identically distributed (non-IID). These factors all affect the performance of models trained in a split federated learning framework with only an end-cloud architecture. In addition, it has been found that model aggregation frequency can significantly affect federated learning performance [14]. This inspired us to propose a multi-level split federated learning (multi-level SFL) framework in which the large-scale client model parameters can be initially aggregated at the edge layer, fog layer, or higher levels, compensating for the scalability of traditional SFL in large AIoT systems. Since the parameters of the client can be preliminarily aggregated at multiple levels before being sent to the cloud server, the communication delay between the cloud server and the client is reduced, the processing speed of the central server is improved, and the global model can be trained by receiving the model parameters of the client in a wider range. The current hierarchical federated learning architecture is composed of client–edge–cloud system, which solves the communication efficiency problem of traditional cloud-based federated learning [15]. However, in the scenario of large AIoT systems based on smart cities, it is difficult for hierarchical FL to cover a larger number of IoT devices and be more widely distributed, so this architecture still limits the number of IoT devices that can be accessed. Compared to hierarchical FL, multi-level architectures can receive a wider range of IoT devices and adjust the number of layers of the architecture as the training task or the number of IoT devices changes, thereby enhancing the generalization of federated learning in AIoT systems. The multi-level aggregation architecture also reduces the single point of failure of the cloud server and resolves the problem of the model training being stuck because the client cannot upload the model information in time due to the long transmission distance. The addition of split learning balances the problem of system heterogeneity among clients and enhances the scalability of the system to incorporate more IoT devices for collaborative learning. Allocating only a portion of the network for training on the end devices can reduce the processing load compared to running a full network in multi-level FL. This enables resource-limited IoT devices to participate in collaborative training, enhancing the diversity of trainable tasks and reducing the impact of data silos in resource-limited devices. Compared with traditional split federated learning, our proposed framework can better solve the system and statistical heterogeneity and

improve the scalability of large-scale AIoT systems. The main contributions of this paper are as follows:

1. This paper proposes a multi-level split federated learning architecture based on IoT device location model information aggregation. The architecture reduces the communication delay between the client and the cloud server. Compared to hierarchical FL, multi-level SFL improves the scalability of the AIoT system through initial aggregation in multi-level edge nodes before the cloud server's aggregation.
2. The split learning algorithm is added to multi-level federated learning, which reduces the impact of system heterogeneity on client collaborative learning and the possibility of abandonment due to limited client computing resources.
3. We utilize the Message Queuing Telemetry Transport (MQTT) protocol to aggregate geographically located IoT devices by sending topics and assigning the nearest master server for split learning training. The client groups in each region communicate with the primary server through their respective local networks.
4. Simulation experiments on multi-level split federated learning using Docker verify that our proposed framework can effectively improve the model accuracy of collaborative training under large-scale clients. In addition, compared with traditional SFL, multi-level SFL in non-IID scenarios can converge faster and reduce the influence of non-IID data on model accuracy.

The rest of this article is organized as follows. Section 2 reports some work related to the article. Section 3 discusses the proposed multi-level split federated learning architecture and associated workflows. In Section 4, the results of the simulation experiments are presented and analyzed. Finally, Section 5 summarizes the article.

2. Related Works

Our proposed work is concerned with three primary DCML topics: multi-level FL, SL, and SFL. Federated learning is an emerging distributed machine learning paradigm that allows clients to jointly model without sharing data, breaking down data silos. With the deepening of research, systematic heterogeneity and statistical heterogeneity have become obstacles to the development of federated learning [16–19]. Karimireddy et al. [20] proposed that the Scaffold algorithm corrects local model updates by adding variance reduction techniques to local training to approximately correct the drift of local training on the client side. In addition to optimizing model parameter aggregation algorithms, there is a lot of work to solve the above problems through personalization techniques [21]. Xu and Fan [22] proposed FedDK, which utilized knowledge distillation for model parameter transmission, and designed the personalized model for each group by using the missing common knowledge to fill circularly between clients. Traditional federated learning frameworks also lack scalability in large AIoT systems. Guo et al. designed [23] a multi-level federated learning mechanism to improve the efficiency of federated learning in device-heavy edge network scenarios by utilizing reinforcement learning techniques to select IoT devices for collaborative training. Campolo et al. [24] proposed a federated learning framework based on the MQTT protocol and lightweight machine-to-machine semantics (LwM2M) to improve communication efficiency and scalability by optimizing message transmission. Furthermore, Liu et al. [25] proposed a multi-level federated learning framework, MFL, which combines the advantages of edge-based federated learning to achieve a balance between communication cost and computational performance for intelligent traffic flow prediction. Multi-level federated learning can also be combined with personalization techniques to solve the problem of statistical heterogeneity. Wu et al. [26] proposed a framework that combines self-attention personalization techniques with multi-level federated graph learning to further capture features of large received signal strength (RSS) datasets for indoor fingerprint localization.

Split learning (SL) is a distributed learning paradigm for resource-limited devices that has the same data-sharing constraint as FL. The principle is to split the deep learning network: each device retains only one part of the network, while the server computes

another part of the network, and the different devices only carry out forward and backpropagation to the local network architecture [27]. SL solves the problem of limited computing resources of edge devices in FL but also increases the communication overhead of the system. Chen et al. [10] proposed a loss-based asynchronous training framework for split learning, which allows the client model parameters to be updated according to the loss, thus reducing the communication frequency of split learning. Moreover, Ayad et al. [28] introduced autoencoders and adaptive threshold mechanisms to track gradients in split learning to reduce the amount of data sent to the client in forward computation and the number of updates in post-feedback communication. Therefore, hybrid split federated learning (SFL) is proposed, which combines the advantages of FL and SL, reduces the communication overhead of split learning, and is suitable for IoT devices with limited resources. In the framework, each client sends its own cutting layer to the master server, which trains the split network and sends the fed server to aggregate the gradient of the split model from each client [29]. Tian et al. [30] split the BERT model according to the calculated load of the embedded layer and transformer layer of the BERT model under the FedBert framework so that it could be deployed on devices with limited resources. Moreover, FedSyL [31], HSFL [32], and ARES [33] frameworks optimized the splitting strategy of deep learning networks, which select the splitting points that can minimize the training cost per round through adaptive analysis based on client computing resources.

Although the above studies have made advances in multi-level federated learning frameworks and split federated learning, they do not take into account collaborative learning in large-scale AIoT system scenarios. Our work greatly improves the scalability of large-scale AIoT systems in collaborative learning by combining the advantages of multi-level federated learning frameworks and the principles of split federated learning.

3. Proposed Framework

In this section, we will introduce multi-level split federated learning in detail, where we group clients according to their geographic distribution and assign corresponding master nodes for model training for split learning, as shown in Figure 1.

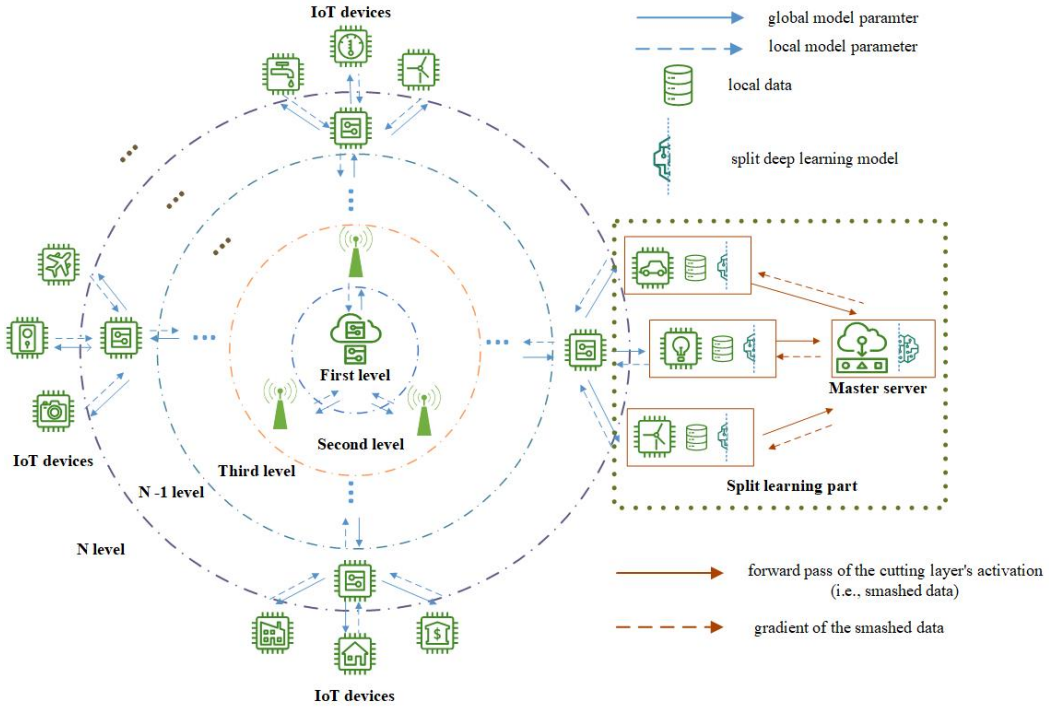

Figure 1. Multi-level split federated learning framework in large-scale AIoT system.

Multi-level split federated learning is designed with a combination of cloud- and edge-based FL and master server-based SL. The cloud server is located in the first level, and the outer level extends to the end device layer of the Nth level, which can contain the $N-2$ level edge servers and fog nodes for the initial aggregation of the local model of the IoT devices. The Nth level is the device layer (end layer), which contains a large number of smart devices and sensors in the smart city, including street lights, sensors, cameras, etc., responsible for sensing the surrounding environment and generating data in real time. The number of levels N of the architecture is determined by the number of IoT devices participating in collaborative training and the area they are located in. As the number of IoT devices increases, the rate at which the cloud server receives model parameters also decreases, increasing the time cost of the computing of the cloud server to some extent. In addition, the larger the range of IoT devices involved in training, the more uneven the transmission distance between different IoT devices, and cloud servers also lead to a reduction in training efficiency. Depending on the number of IoT devices and their geographic location required for different training tasks, the proposed architecture automatically adds adjacent and idle edge nodes or fog or edge nodes with more computing power for hierarchical expansion. Since the infrastructure of most smart cities is fixed, we assume that in this design, the IoT devices will be stable when training collaboratively, and the geographical location will be constant during training time. The IoT nodes in the end layer (Nth level) are composed of heterogeneous resource-constrained devices, the data generated by each IoT node are saved locally and cannot be shared, and each uses its own local data trainer for the local model. Depending on the geographical location of the IoT devices, they are divided into different groups, and each group is assigned the nearest master server, which is the server with high-performance computing resources. All IoT devices in parallel perform forward propagation on their local model and transmit their smashed data to the master server. The master server calculates the forward propagation and backward propagation of the smashed data for each IoT device in its server-side model and sends the gradient to the respective IoT devices to operate backpropagation. The closer the edge nodes of each layer of the multi-level SFL are to the cloud server, the stronger the computing and communication capabilities. For example, edge nodes such as routers and gateways close to IoT devices have stronger computing performance and storage space than IoT devices. Higher-level edge nodes or fog nodes that connect to this level, such as base stations or regional servers, have more computing capabilities than edge nodes closer to IoT devices. Therefore, the highest level (the first level) consists of one cloud server. After the edge nodes of each layer initially aggregate the parameters of the local model of the IoT device through the FedAvg algorithm, the aggregated model parameters are further uploaded to the cloud server for the aggregation of global model parameters. Compared with the traditional SFL architecture, multi-level SFL does not need to wait for all clients to update and upload model parameters to a single cloud server for parameter aggregation, reducing the communication cost of AIoT system collaborative learning and expanding the IoT devices available for training. The detailed workflow of multi-level SFL is as follows.

3.1. MQTT Protocol-Based Message Exchange

Message Queuing Telemetry Transport (MQTT) is a lightweight communication protocol based on the publish/subscribe model, which is built on the TCP/IP protocol and commonly used in Internet of Things systems. The advantage of MQTT is that it provides reliable messaging services for connecting remote devices with limited bandwidth, so it is suitable for low-bandwidth environments consisting of resource-limited end devices. In the multi-level SFL framework, we use the MQTT protocol for message exchange between client nodes and server nodes. The server of MQTT is brokered by VerneMQ. Edge nodes and device nodes find device nodes that split federated learning on the MQTT topic and control the aggregation operation of model parameters. The specific complete topic is shown in Table 1.

Table 1. Topics used in the MQTT protocol.

Message Exchange.	Topic	Details
Device–Edge	client/group	IoT devices are grouped according to the public socket IP they provide, and the corresponding master server is assigned.
	train/start	The grouped IoT devices receive the signal from the edge server to start the model training.
	train/update	When the server needs to receive the local model weight of the IoT device, the IoT device receives the signal from the edge server.
Edge–Edge Edge–Cloud	train/start	The edge server receives the signal from the upper-level server and starts to trigger the model training.
	train/update	At the beginning of each communication turn, the selected clients are notified that their weights will be aggregated.
	client/join	Receive messages when a new IoT device joins a group. The message includes contextual information about the newly joining IoT device, including its status (whether it can participate in training) and its public socket IP.

The MQTT agent publishes the subject client/group, groups IoT devices under the same LAN by receiving the IP of each IoT device, and assigns the master server node under the same LAN. The server nodes on the connected edge of each group connect to the upper-layer edge nodes or cloud servers through the backbone (internet) network. At the beginning of each round of communication, the edge server will randomly aggregate the model parameters of the IoT devices by publishing a list of IDs containing the selected IoT devices to the train/update topic. When an IoT device receives a message published on this topic, it checks if its ID is in the list, and if so, it sends its local model parameters to the edge node; otherwise, it receives the average weight of the model parameters from the edge node. The message exchange between edge nodes at each layer and between edge nodes and cloud servers is similar. However, when the new IoT device node joins the group, it receives the message of the new IoT device through the topic client/join, and groups and evaluates the new IoT devices according to their own information.

To analyze the communication overhead of the proposed framework through MQTT, we assume M represents the number of parameters for the model, D represents the total number of IoT devices, s represents the size of the total samples, q is the size of the smashed layer, E represents the total number of edge nodes in the middle levels, and α represents the proportion of model parameters on the IoT device side. For instance, the model parameters in IoT devices can be expressed as αM, and in the master server, they can be expressed as $(1-\alpha)M$. Multi-level SFL mainly carries out two parts of information transferred through MQTT protocol. The first part is the transfer of information between the IoT device and the master server. The information that the IoT device transfers to the master server is the smash data from the cutting layer of the local model, and the information that the master server transfers to the IoT device is the gradient of the smash data. Therefore, the size of the information transmitted by the split learning section depends on the size of the private data generated by the local IoT device. The total communication overhead in the split learning part can be denoted as $2sq$. The second part of the communication overhead comes from the size of the information transmitted by multi-level federated learning. The information transferred size of FL depends on the number of parameters in the IoT device's local model [34], but since the local model is split, the communication overhead of multi-level FL can be expressed as $2\alpha M(D+E)$. Therefore, the communication overhead of a multi-level SFL can be expressed as $2sq + 2\alpha M(D+E)$. Table 2 shows a comparison of the size of information transferred by the proposed framework over MQTT versus that transferred by traditional federated learning. As can be seen from Table 2, multi-level SFL can reduce the size of information transmitted by the device and improve the transmission efficiency of the framework when training large deep learning models with the same size of training data.

Table 2. Communication overhead of different distributed learning methods.

Method	Comms. per IoT Devices	Total Comms.
FL	$2M$	$2DM$
SL	$(2s/D)q$	$2sq$
Multi-level SFL	$(2s/D)q + 2\alpha M$	$2sq + 2\alpha M(D + E)$

3.2. Split Learning Side

After grouping IoT devices according to their geographic location, each group first performs a split learning algorithm with the assigned master server. The algorithm can be divided into four main computational parts. The IoT device is responsible for performing two parts of the deep learning network computation, namely the forward propagation and backward propagation of the IoT device network. The master server is responsible for calculating the remaining layers and loss calculations. We define a deep learning network as a function f, which contains the network layers that can be represented as $\{l_0, l_1, \ldots, l_K\}$. For the input local data $(local\ data)$, the output of the neural network is

$$l_K(l_{K-1} \ldots (l_0(local\ data))) \rightarrow f(local\ data). \tag{1}$$

Let $Loss(output, label)$ represent the last layer to calculate the loss function of the real label and the network output. ${\Delta l_i}^T$ represents the backpropagation process at each network layer, so the backpropagation of the entire deep learning network can be expressed as

$${\Delta l_K}^T \left({\Delta l_{K-1}}^T \ldots \left({\Delta l_i}^T \right) \right) \rightarrow \Delta f^T. \tag{2}$$

Therefore, the deep network structure can be divided into two parts according to the disassembly layer, assuming that the layer l_n is the cutting layer of the neural network. The network layer retained by the IoT device and the master server node is represented as follows

$$f_c \leftarrow \{l_0, l_1, \ldots, l_n\}, \ \ n \in N, \tag{3}$$

$$f_m \leftarrow \{l_{n+1}, l_{n+2}, \ldots, l_N\}, \ \ n \in N. \tag{4}$$

The client sends the activation of the cutting layer generated by the forward propagation of the local network model to the master server for forward propagation of the rest of the network layers. The master server is responsible for calculating the loss of labels and outputs and propagating backward to update the master server's network layer weights. The client smashed data from the forward propagation of the local network model to the main server for forward propagation of the rest of the network Assuming a set of D IoT devices in t time period, the model update of the master server part of the network can be expressed as follows [29]:

$$W_{t+1}^M := W_t^M - \eta \frac{s_D}{s} \sum_{m=1}^{D} \Delta L_m \left(W_t^M; \alpha_t^M \right), \tag{5}$$

where η is the learning rate to train the deep learning model, and s is the size of the total samples. α_t^M represents the activation in master server, and $\Delta L_D \left(W_t^M; \alpha_t^M \right)$ denotes the gradient of backpropagation in the master server's network layer.

Algorithm 1 shows the exact algorithm flow of the split learning part. The backpropagation gradient received by the client from the master server is sent to the edge server for aggregation, so Algorithm 1 is only one part of the multi-level SFL, and the entire algorithm will be presented later.

Algorithm 1 Split learning part in multi-level split federated learning

Notations: s is the size of total samples; t is time period; α_t^d is the smashed data of IoT device at t; ΔL_d is the gradient of the loss for IoT device d; Y_d is the true label from IoT device d.
Initialize: for each IoT device $d \in D$ in parallel **do**
$f_d \leftarrow \{l_0, l_1, \ldots, l_n\}$, initialize weight f_d using W_t^d
end for
In master server: $f_m \leftarrow \{l_{n+1}, l_{n+2}, \ldots, l_N\}$, initialize weight f_m using W_t^M

1: **for** each IoT device in the same group $d \in D$ in parallel **do**
2: **while** local epoch $e \neq E$ **do**
3: IoT device received model weight the from cloud server
4: Forward propagation compute activation (smashed data) on cutting layer $\alpha_t^d \leftarrow f_d(data)$
5: Send activation α_t^d and local label Y_d to master server
6: **end while**
7: **Master server executes:**
8: Forward propagation with α_t^d on f_m, compute $output \leftarrow f_m\left(\alpha_t^d\right)$
9: Calculate Loss $L_d \leftarrow lossfunction(output,\ Y_d)$
10: Backpropagation on f_m, calculate the gradient $\Delta L_d\left(W_t^M; \alpha_t^M\right)$
11: Send gradient of cutting layer $d\alpha_t^d := \Delta L_d\left(W_t^M; \alpha_t^M\right)$ to IoT device for backpropagation
$f_d \leftarrow backPropagation\left(W_t^d,\ d\alpha_t^M\right)$
12: **end for**
13: Model f_m from master server update $W_{t+1}^M := W_t^M - \eta \frac{s_d}{s} \sum_{m=1}^{d} \Delta L_m\left(W_t^M; \alpha_t^M\right)$

3.3. Multi-Level Federated Learning Workflow

The cloud server publishes topics through the MQTT protocol to match the edge server to the nearest IoT device group. Then, the edge server publishes topics to receive the tasks performed by the IoT device, information about the local model, and information about the respective resources of the IoT device, including computing resources and storage resources. Based on the IoT device information, the edge server will determine whether the IoT device meets the requirements of collaborative training (whether it is idle and has enough computing resources) and send the training task to the cloud server. Depending on the training task, the cloud server randomly initializes the global model parameters and sends them to the IoT device. Each group of IoT devices receives the cutting layer gradient from the master server and backpropagates the local model to update the parameters of the neural network. The next major step is parameter aggregation in the edge server and cloud server, where we use the FedAvg aggregation algorithm.

After the IoT devices in each group update the weights of the local model through the model gradient sent by the master server, they send the model parameters (weights) to the associated edge server. Each edge node is responsible for collecting updated parameters from local IoT devices in its region and using the FedAvg algorithm for weighted averaging to update the weights of the local partial model (initial aggregation). Suppose there are E edge servers; then, the FedAvg algorithm [35] can be expressed as

$$W^e = \sum_{m=1}^{D} \frac{s_m}{s} W_d^e, \quad e \in E. \tag{6}$$

There are two situations when model parameters are sent to an edge server. If the edge device is in the middle layer of the hierarchy, the aggregation of model parameters can be expressed as

$$W^{e_{N-1}}{}_{t+1} = \frac{\sum_{d_i \in e_{N-1}} S_d W_t^d}{S_D^{e_{N-1}}}, \quad e \in E, \tag{7}$$

where $S_d \epsilon S_D$, S_D denotes the size of the total samples in the IoT device from one group under the e_{N-1}, which is the edge server located in layer $N-1$. The edge node in the middle layer will forward the aggregated average weight to the upper connected edge

node to aggregate again. Similarly, the edge server in level $N-2$ executes the aggregation algorithm and can be represented as

$$W^{e_{N-2}}{}_{t+1} = \frac{\sum_{d_i \in e_{N-2}} S_{e_{N-1}} W^{d}_{e_{N-1},t}}{S_D^{e_{N-2}}}, \quad e \in E. \tag{8}$$

If the edge node is at the top of the hierarchy, that is, the cloud server, the weights from the edge nodes of the previous layer are aggregated and weighted and then sent back to the lower level. The cloud server aggregates all the model parameters for each group of IoT devices. The aggregation operation in the server can be expressed as

$$W^{C}{}_{t+1} = \frac{\sum_{S_{e_2^d} \in S} S_{e_2^d} W^{d}_{e_2,t}}{S_D^{e_{N-2}}}, \quad e \in E. \tag{9}$$

The final average weight is sent from the cloud server back to the edge server and client in the same path as the model parameters were previously uploaded, as shown in Figure 2. After receiving the global parameters, each set of IoT devices updates its local model and forwards the cutting layer activation to the master server for forward propagation and loss calculation. The entire multi-level federation learning begins a new iteration until the model converges. Algorithm 2 illustrates the precise algorithm flow of the multi-level SFL framework and shows the algorithm of the multi-level FL part in detail.

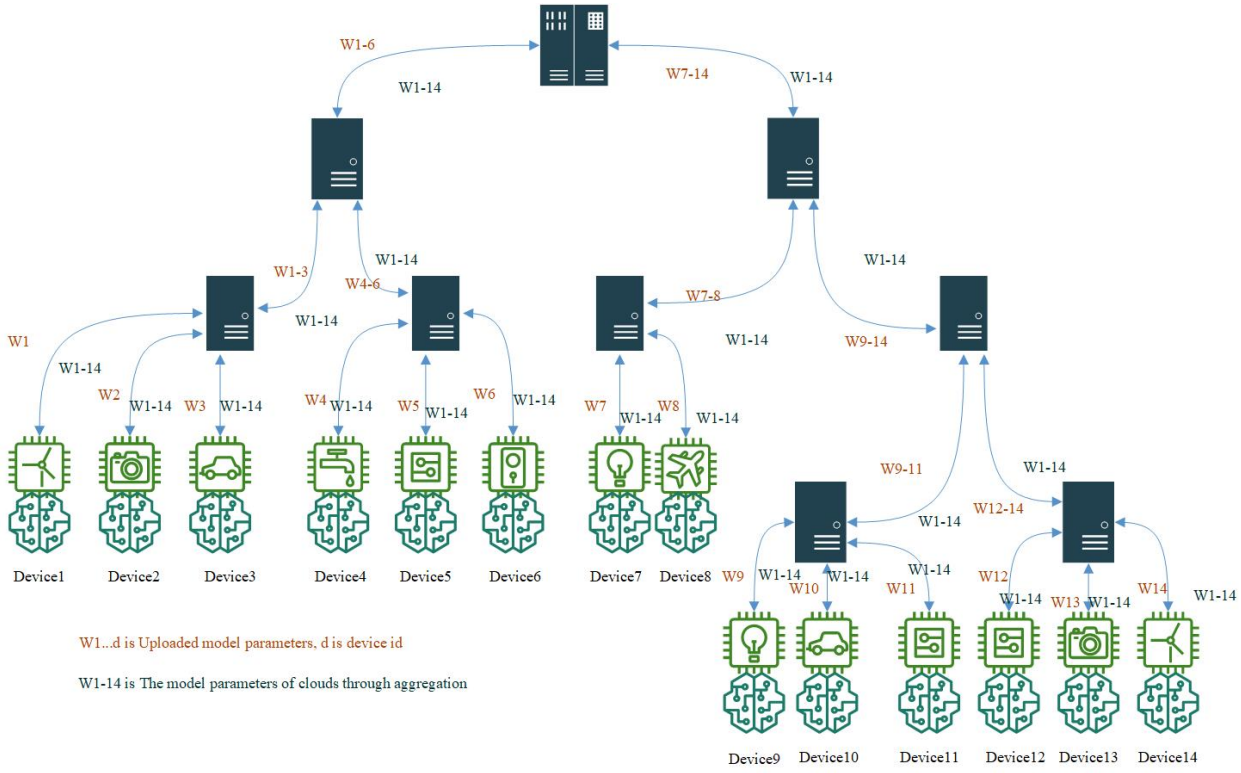

Figure 2. Multi–level federated learning workflow.

Algorithm 2 Multi-level SFL algorithm and multi-level FL workflow

Notations: s is the size of total samples; t is time period; E is the number of edge servers in each level.
Initialize: $level = N$; global model in cloud server W^C;

if t = 0 **do**
send W^C to all IoT devices for model weight initialization
else
master server and IoT device executes split learning part,
$f_d \leftarrow backPropagation\left(W_t^d,\ d\alpha_t^M\right)$
if $level = N - 1$ **do**
for each IoT device $S_d \epsilon S_D$ in parallel **do**
Send model weight to edge sever $W_{t+1}^E \leftarrow backPropagation\left(W_t^d,\ d\alpha_t^M\right)$
end for
Aggregate model parameter $W^{e_{N-1}}{}_{t+1} = \frac{\sum_{d_i \in e_{N-1}} S_d W_t^d}{S_D^{e_{N-1}}}$
end if
level --;
if $level \neq 1$ **do**
for edge server $e \in E$ **do**
Send model weight to upper-level edge sever $W^{e_{N-1}}{}_t \leftarrow W^{e_{N-2}}{}_t$
end for
Aggregate model parameter utilized FedAvg algorithm
level --;
end if
if $level = 1$ **do**
for edge server $e \in E$ **do**
Send model weight to cloud sever
end for
Aggregate model parameter from lower-level edge server (all IoT devices model parameter)
$W^C{}_{t+1} = \frac{\sum_{S_{e_2^d} \in S} S_{e_2^d} W_{e_2^d,t}^d}{S_D^{e_{N-2}}}$
Send $W^C{}_{t+1}$ to all lower-level edge server and all IoT device as previous upload path
end if
end if

4. Experiment

In this section, we describe the performance of multi-level split federated learning on different datasets (Fashion MNIST, HAM10000) and different machine learning models (LeNet, ReNet18). The feasibility of our proposed framework is verified by comparing the traditional split federated learning, such as SFLV1, multi-level federated learning, and centralized learning in an independent identically distributed (IID) and balanced dataset, an unbalanced dataset, and a non-independent identically distributed (non-IID) dataset. We also tested the performance of multi-level SFL with a different number of clients and demonstrated that multi-level architecture can reduce the impact of an increasing number of clients on model training accuracy. Since the aim of the experiments is to simulate the real smart cities AIoT system in Docker, the model accuracy and the time cost of training the model will be the evaluation criteria to test the framework we proposed.

4.1. Experiment Setting

The experiment is built on Docker 24.0.7 and API 1.43, using multiple isolated Docker containers to simulate the end devices in the smart city AIoT system, such as cameras, indicators, temperature sensors, etc. Docker compose v2.23.3 is used to manage multiple clusters of Docker containers. To simulate a close-to-real-world scenario, clients have been assumed to be divided into distinct groups based on their region. Due to each region having its own WLAN or PAN, the clients in each group are connected through Docker's

bridge network built under different IP addresses (local network). The upper-level nodes of each group, namely edge nodes and fog nodes, are connected to the cloud server through a global network. The MQTT protocol is used for message transfer between nodes at various levels, through the publication and subscription of messages to manage the joining of group nodes, the start and end of model training, and the aggregation of model parameters. We also use Secure Sockets Layer (SSL) sockets to add a secret key to the transmission of model weights in the system, so the sent weights will be hashed together with the key, ensuring the security and privacy of data transmission in the real world. The experiment used SSL single authentication, that is, the client should authenticate the identity of the cloud server, and the server does not need to authenticate the client. Once the authentication is complete and the server and client SSL session is established, the two parties begin an MQTT connection over the secure SSL channel and communicate over the encrypted channel by publishing and subscribing context, as shown in Figure 3.

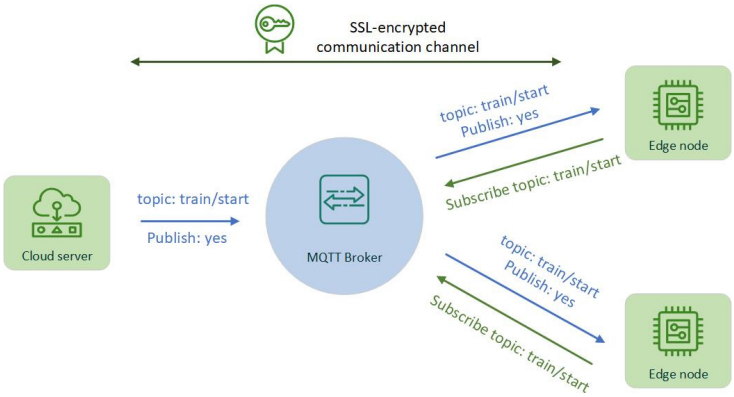

Figure 3. Experiment simulation smart cities MQTT protocol with encrypted SSL.

All programs are written by Python3.6 and TensorFlow1.8 and built on a Windows computer with an NVIDIA GeForce RTX4090 GPU (Santa Clara, CA, USA) and Intel Core i9-12900K CPU (Santa Clara, CA, USA). The NVIDA GeForce RTX4090 is a public version of the card manufactured by NVIDIA, and the device is sourced from the United States. Intel Core i9-12900K CPU is Boxed Intel® Core™ i9-12900K Processor (30M Cache, up to 5.20 GHz) FC-LGA16A, for China. We selected HAM10000 and Fashion MNIST datasets, ResNet18 and LeNet, as machine learning network architectures to train these two datasets, respectively. Both architectures belong to the classical convolutional neural network-type architectures. ResNet18, consisting of 15 convolutional layers and 2 pooling layers, was used to test the proposed framework as a large machine learning task. Moreover, LeNet contains three convolutional layers and 2 pooling layers as machine small learning tasks to test the proposed framework. In addition, the learning rate of both networks is 0.0001.

In all experiments under the split federated setting, the network is split according to the following layer: the third layer of ResNet18 (the BatchNormalization layer) and the second layer of LeNet (the MaxPool layer). Generally, two factors are considered in the selection of the cutting layer of the machine learning model: one is the proportion of computing amount between the end device and the master computing node after the model is split, and the other is the hidden layer feature dimension of the cutting layer. The former is mainly determined by the processing speed, memory size, and power consumption budget of the end device. The latter is mainly determined by the bandwidth of the network connection between the end device and the master server. Since the experiment is set for training on resource-limited IoT devices and the bandwidth of MQTT protocol is much less than other protocols [36], the choice of destratification of the model does not need to consider the size of the feature dimension too much but rather the computing resources of the IoT devices. Therefore, we chose to train on IoT devices at a model layer with less split while ensuring that fragmented data do not compromise the privacy of the source data.

4.2. Experiment Dataset and Simulation

Two public image datasets were used in the experiment: Fashion MNIST [37] and HAM10000 [38], as shown in Table 3. Fashion MNIST is an image dataset that replaces the MNIST handwritten numerals set. It includes front images of 70,000 different products from 10 categories. Fashion MNIST's size, format, and training/test set division are exactly the same as the original MNIST, with a 60,000/10,000 training test data partition, 28 × 28 gray scale picture. As the MNIST dataset is too simple and the amount of data is small, Fashion MNIST is more consistent with the machine learning tests AIoT system based on smart cities. The HAM10000 dataset consists of 10,015 dermatoscopic images for the classification of pigmented skin lesions. There are seven labels: Akiec, bcc, bkl, df, mel, nv, and vasc. The number of samples in each category of the HAM10000 dataset is not the same, so its sample imbalance is prone to overfitting. We used this dataset to test the performance of multi-level SFL under unbalanced samples. The dataset is divided by the number of clients, each client holds a portion of the dataset, and the label of the dataset is stored in the master server of each client group.

Table 3. Training and testing of dataset.

Dataset	Training Samples	Testing Samples	Image Size
Fashion MNIST [37]	60,000	10,000	28 × 28
HAM10000 [38]	9013	1002	600 × 450

In order to simulate the limited computing resources of the IoT device and the sufficient computing resources of the master server used for computing in split learning, part of the model trained in the client is calculated by the CPU, while the other part of the model in the server is calculated by the GPU. Since the cloud server only needs to perform the task of parameter averaging, the cloud server is also simulated by a container in Docker. Cloud server containers publish task topics to edge nodes over the backbone network, and multiple containers simulate different groups of IoT devices receiving task selection datasets from edge nodes. The containers of each group are connected via a local network within the group, but each container trains the local model independently and communicates only when a topic is published for new devices to join. MQTT protocol is simulated by docker-vernemq, and container clusters are deployed by docker-composer. All containers are independent of each other to simulate the condition that data cannot be shared between IoT devices.

4.3. Performance of Multi-Level SFL, FL, and Centralized Learning

Centralized learning and multi-level federated learning serve as benchmarks for testing our proposed multi-level split federated learning. Multi-level federated learning and multi-level split learning are, respectively, tested in the AIoT system of four levels. The end layer is set with 50 nodes, which are divided into two groups according to geographical location, and the edge layer and fog layer are set with 2 nodes, which are, respectively, responsible for aggregating the model weights of the two groups of end nodes. The cloud has one node responsible for performing the FedAvg algorithm to aggregate the model weights of the edge nodes. Table 1 summarizes the accuracy of distributed collaborative learning over 50 global epochs with a batch size of 32 (Fashion MNIST) or 1024 (HAM10000) for each local epoch.

As shown in Table 4, multi-level split federated learning and multi-level federated learning perform well in experimental settings, and there is no significant difference between centralized learning and multi-level federated learning. Although centralized learning on both machine large learning tasks (ResNet18) and small machine learning tasks LeNet) has slightly better convergence results than multi-level split federated learning and federated learning, only a few accuracy losses are negligible. Moreover, we compare multi-level SFL and FL, both of which do not overfit on the HAM10000 and Fashion

MNIST datasets, because multi-level SFL and FL execute the FedAvg algorithm on multiple upper layers, so the client can reduce the number of local model updates. In addition, as shown in Table 4, multi-level SFL performs better than multi-level FL in the HAM10000 dataset. Because the sample size of the HAM10000 dataset is unbalanced and multi-level SFL inherits the properties of SL, machine learning performance is better under the unbalanced dataset.

Table 4. Train and test result in different distributed setting.

Dataset	Architecture	Centralized Learning		Multi-Level FL		Multi-Level SFL	
		Train	Test	Train	Test	Train	Test
HAM10000	ResNet18	74.4%	79.6%	76.9%	77.3%	78.6%	79.4%
Fashion MNIST	LeNet	88.7%	90.2%	86.1%	87.6%	87.9%	88.9%

In Figure 4, the X-axis represents the epoch of training, the Y-axis on the left represents the accuracy of training, and the Y-axis on the right represents the training loss. It can be seen from the experimental results that both multi-level SFL and FL can converge after 50 epochs on the Fashion MNIST dataset, reaching an accuracy of 88.9% and 88.6%. We note that multi-level SFL and FL converge first, while centralized learning begins to converge later, presumably because the dataset each client trains on is too small and the data type is large due to the random allocation of the dataset. Furthermore, the convergence rate of multi-level SFL is faster, and the model began to converge at about 20 epochs. This shows that multi-level SFL can reduce the communication overhead during training.

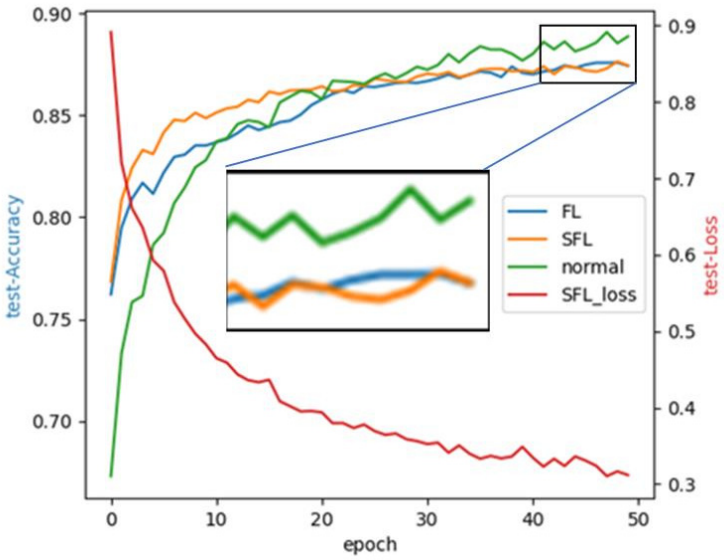

Figure 4. Testing convergence of LeNet on Fashion MNIST under various learning.

4.4. Effect of Different Clients on Performance

In this section, we will analyze the influence of edge client data on model training accuracy. For multi-level SFL, we increase the number of nodes on each tier as well as the number of clients for aggregation of model weights, as shown in Table 5. Considering that the master server running the split model needs to run the split model for each client node, we assume that each client group in different regions is equipped with a master server node, so during the model training and optimization process, we do not consider the transfer time between the master service node and the client node.

Table 5. Number of clients and multi-level nodes for training.

The number of clients	5	10	20	50
The number of edge nodes in edge level	2	2	2	4
The number of fog nodes in fog level	1	1	2	2
The number of cloud servers	1	1	1	1
The number of master servers	2	2	2	4

This section will analyze the impact of the number of LeNet users on Fashion MNIST. Figure 5 shows how the test accuracy and convergence rate vary with the number of epochs when a multi-level SFL is trained on a different number of clients (5, 10, 20, 50 clients). We can see that as the number of clients increases, the convergence speed of multi-level SFL will slow down, but the convergence speed is not obvious, and it is always better than the convergence speed of centralized learning. Moreover, for multi-level SFL, presumably, as the number of clients increases, the model accuracy decreases. For example, when the number of clients reaches 20 and 50, the accuracy of the test is significantly lower than that of the concentrated learning. However, the accuracy of the model trained on 50 clients is slightly higher than the accuracy of the model trained on 20 clients because the edge nodes of the middle layer also increase as the layers of the client are interlayer, so the number of middle-layer nodes of the multi-level SFL helps the SFL reduce the loss of accuracy.

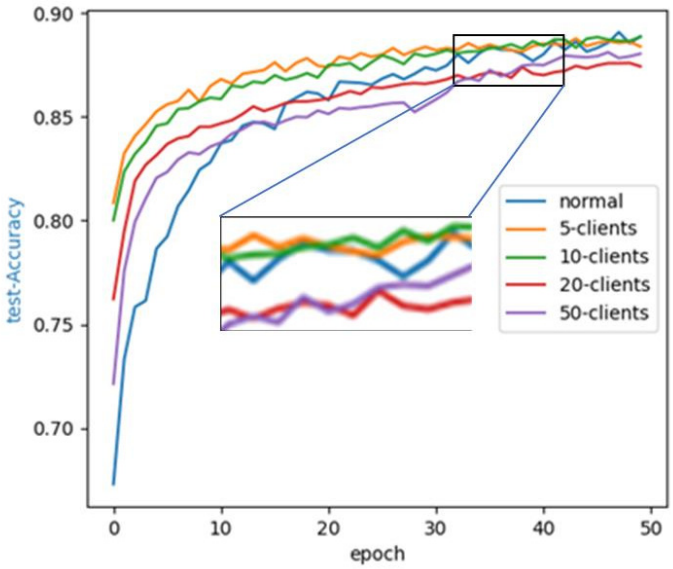

Figure 5. Effect of the number of clients on testing accuracy for LeNet on Fashion MNIST.

Figure 6 shows the time cost required for LeNet to train the Fashion MNIST dataset with a different number of clients. As can be seen from the figure, the training time required for centralized learning is always better than the time cost of training the model under multi-level SFL with different clients. This is because the experiment simulates the MQTT protocol used by the AIoT system to communicate, so the communication time and the time spent waiting for all the clients to train will layer the time cost of the multi-level SFL. The training time of the model decreases first and then increases with the increase in clients. This is because as the number of multi-level SFL clients increases, the amount of data locally decreases accordingly, reaching the minimum time overhead with 20 clients. However, when the number of clients reaches 50, the time overhead for model training starts to rise because the time for cloud servers and nodes in the middle level to publish topics and receive requests under the MQTT protocol starts to increase.

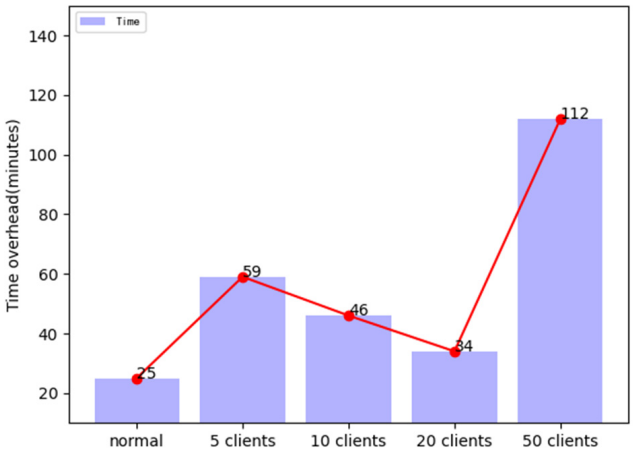

Figure 6. Time required for LeNet model to reach convergence in number of clients.

4.5. Impact of Different Level Layer on the SFL Model Training

Traditional split federated learning architecture, such as SFLV1 and SFLV2 [29], has only two levels: the server executing FedAvg and the clients with the master server. It has only one operation of model parameter aggregation, and when a client is corrupted, the FedAvg server will wait and freeze. Multi-level SFL performs model aggregation operations at multiple levels, and due to the MQTT protocol, nodes performing aggregation operations can communicate with clients by publishing topics, so there is no need to wait for broken nodes. Therefore, we set up two SFL scenarios: traditional SFL (SFLV1) and multi-level SFL, both of which have 20 clients for collaborative learning, and multi-level SFL with 4 edge nodes and 2 fog nodes to perform aggregation of global model weights. In addition, we make the dataset to which each client is assigned non-independent and identically distributed (non-IID). Moreover, the local dataset of the clients is divided according to the label distribution of the sample, which means the sample label distribution on each client is different.

Figure 7 shows the relationship between epochs and model test accuracy by non-IID SFLV1 and multi-level SFL architectures under the MQTT communication protocol. After 50 epochs, multi-level SFL and SFLV1-trained LeNet models can achieve approximately 88% and 86% accuracy, respectively, under non-IID. As can be seen from the figure, in the non-IID scenario, multi-level SFL is superior to traditional SFL, with faster convergence and higher model accuracy. This indicates that non-IID has a negative effect on splitting federated learning, but multi-level SFL can improve this problem. In the federated learning part, compared with traditional split federation learning, multi-level split federation learning can aggregate more clients at the same time for model training, so the model convergence speed will be faster. In the split learning part, since the multi-level SFL performs aggregation operations at each layer, the multi-level SFL clients undergo more local model updates, which alleviates the non-IID problem.

4.6. Comparison of Multi-Level SFL and FL Time Cost

This section compares the time overhead of multi-level SFL and multi-level FL when training large (ResNet18) and small (LeNet) machine learning tasks. Both use a four-level multi-level architecture, and the number of clients is 20. In order to simulate the limited computing resources of the client and the efficient computing resources of the master server, the local training of the client uses the CPU for computing, while the split model trained by the master server uses the GPU for training. We test the impact of multiple levels of SFL on resource-limited clients by the time overhead on model training.

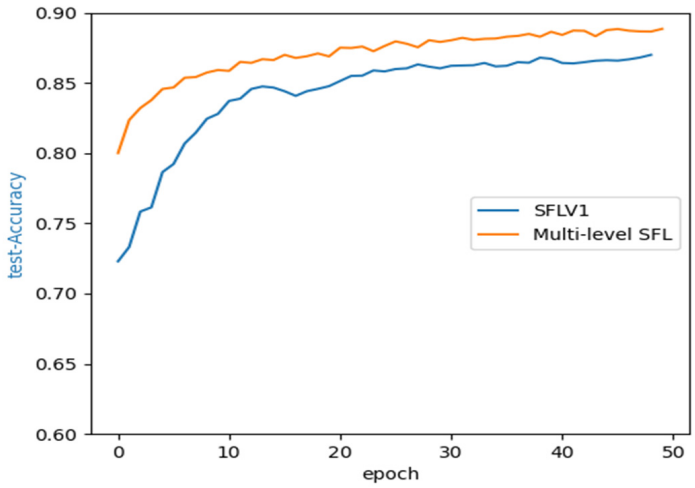

Figure 7. Testing convergence of LeNet on non-IID dataset.

As shown in Figure 8, centralized learning has the lowest time cost for both small and large machine learning tasks, at 25.32 min and 85.57 min, respectively. In the training of small model LeNet, the time cost of multi-level FL is slightly lower than that of multi-level SFL, and the difference in time cost between the two is not obvious. This is because when training the small network LeNet, the computing cost of the two-layer network in the local training of the client is relatively small, and the master server only needs to train the three-layer network, so the impact of split learning on the global training of the AIoT system cannot be reflected. Furthermore, because the master server and client also communicate via the MQTT protocol, multi-level SFL has a partial longer topic release time than multi-level FL. Therefore, in the small network training, multi-level SFL cannot present its advantage. However, on a large network, such as ResNet18, which has 18 layers, 15 of which are computed by the master server, the time cost of multi-level SFL is significantly lower than that of multi-level FL. And because the split learning process of multi-level SFL is run in parallel in the main server, rather than a linear run similar to split learning, the time cost of multi-level learning in large machine learning tasks is lower. The experimental results show that multi-layer SFL is more suitable for clients in AIoT with limited resources than multi-layer FL.

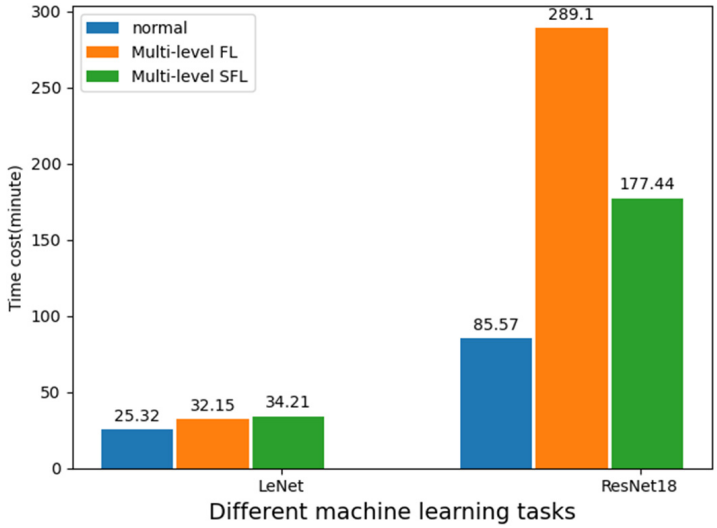

Figure 8. Time cost of multi-level SFL and multi-level FL train different models.

5. Conclusions

In this work, we proposed a novel multi-level split federated learning (SFL) framework for the enhancement of collaborative learning in large-scale AIoT systems. The multi-level SFL framework addresses the connectivity and data processing challenges that occur in a large-scale AIoT system with a multitude of clients. Through multiple levels of aggregation of model parameters, it significantly reduces the communication delay between the cloud server and the clients, enhancing the processing speed and entire performance of the central server. By integrating split learning into the framework, it balances the system heterogeneity among clients and boosts the system's scalability to incorporate more IoT devices for collaborative learning. It also mitigates the single point of failure risk of the central cloud server and ensures continuous model training even in the event of longer transmission distances. The use of the Message Queuing Telemetry Transport (MQTT) protocol and Docker containers in the experimental setup substantiates the practical feasibility of the proposed multi-level SFL framework. The resulting improvements in model accuracy under large-scale clients and faster convergence in non-IID scenarios, as evidenced by the simulation experiments, further validate the effectiveness of the proposed solution. Although our proposed multi-level SFL architecture has shown some advantages in model accuracy, it still has some shortcomings in terms of transmission overhead. For example, we have not further explored the effect of different sized datasets on the size of the information transferred and how to balance the overall communication overhead through MQTT generated by federated learning and split learning. Future work will further study these aspects.

Author Contributions: Conceptualization, H.X., K.P.S. and L.M.A.; methodology, H.X. and K.P.S.; resources, K.P.S.; data curation, H.X., K.P.S. and L.M.A.; writing—original draft preparation, H.X., K.P.S. and J.S.; writing—review and editing, K.P.S., J.S. and L.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The Fashion MNIST dataset can be found in Kaggle: https://www.kaggle.com/datasets/zalando-research/fashionmnist (accessed on 15 January 2024). The HAM10000 dataset comes from the paper: "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions", DOI: https://doi.org/10.1038/sdata.2018.161.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Su, Z.; Wang, Y.; Luan, T.H.; Zhang, N.; Li, F.; Chen, T.; Cao, H. Secure and Efficient Federated Learning for Smart Grid With Edge-Cloud Collaboration. *IEEE Trans. Ind. Inform.* **2022**, *18*, 1333–1344. [CrossRef]
2. Xu, C.; Qu, Y.; Luan, T.H.; Eklund, P.W.; Xiang, Y.; Gao, L. An Efficient and Reliable Asynchronous Federated Learning Scheme for Smart Public Transportation. *IEEE Trans. Veh. Technol.* **2023**, *72*, 6584–6598. [CrossRef]
3. Lian, Z.; Yang, Q.; Wang, W.; Zeng, Q.; Alazab, M.; Zhao, H.; Su, C. DEEP-FEL: Decentralized, Efficient and Privacy-Enhanced Federated Edge Learning for Healthcare Cyber Physical Systems. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 3558–3569. [CrossRef]
4. Taïk, A.; Mlika, Z.; Cherkaoui, S. Clustered Vehicular Federated Learning: Process and Optimization. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 25371–25383. [CrossRef]
5. Bebortta, S.; Tripathy, S.S.; Basheer, S.; Chowdhary, C.L. FedEHR: A Federated Learning Approach towards the Prediction of Heart Diseases in IoT-Based Electronic Health Records. *Diagnostics* **2023**, *13*, 3166. [CrossRef]
6. Hsu, R.-H.; Wang, Y.-C.; Fan, C.-I.; Sun, B.; Ban, T.; Takahashi, T.; Wu, T.-W.; Kao, S.-W. A Privacy-Preserving Federated Learning System for Android Malware Detection Based on Edge Computing. In Proceedings of the 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), Taipei, Taiwan, 20–21 August 2020; pp. 128–136.
7. Yamamoto, F.; Ozawa, S.; Wang, L. eFL-Boost: Efficient Federated Learning for Gradient Boosting Decision Trees. *IEEE Access* **2022**, *10*, 43954–43963. [CrossRef]
8. Jiang, L.; Wang, Y.; Zheng, W.; Jin, C.; Li, Z.; Teo, S.G. LSTMSPLIT: Effective SPLIT Learning Based LSTM on Sequential Time-Series Data. In Proceedings of the 36th AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 28 February–1 March 2022.
9. Hsieh, C.-Y.; Chuang, Y.-C.; Wu, A.-Y. C3-SL: Circular Convolution-Based Batch-Wise Compression for Communication-Efficient Split Learning. In Proceedings of the 2022 IEEE 32nd International Workshop on Machine Learning for Signal Processing (MLSP), Xi'an, China, 22–25 August 2022.

10. Chen, X.; Li, J.; Chakrabarti, C. Communication and Computation Reduction for Split Learning Using Asynchronous Training. In Proceedings of the 2021 IEEE Workshop on Signal Processing Systems (SiPS), Coimbra, Portugal, 19–21 October 2021; pp. 76–81.
11. Abedi, A.; Khan, S.S. FedSL: Federated Split Learning on Distributed Sequential Data in Recurrent Neural Networks. *Multimed. Tools Appl.* **2023**. [CrossRef]
12. Wu, Y.; Kang, Y.; Luo, J.; He, Y.; Yang, Q. FedCG: Leverage Conditional GAN for Protecting Privacy and Maintaining Competitive Performance in Federated Learning. *arXiv* **2022**, arXiv:2111.08211.
13. Zhang, Z.; Pinto, A.; Turina, V.; Esposito, F.; Matta, I. Privacy and Efficiency of Communications in Federated Split Learning. *IEEE Trans. Big Data* **2023**, *9*, 1380–1391. [CrossRef]
14. Deng, Y.; Lyu, F.; Ren, J.; Zhang, Y.; Zhou, Y.; Zhang, Y.; Yang, Y. SHARE: Shaping Data Distribution at Edge for Communication-Efficient Hierarchical Federated Learning. In Proceedings of the 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), Washington, DC, USA, 7–10 July 2021; pp. 24–34.
15. Liu, L.; Zhang, J.; Song, S.H.; Letaief, K.B. Client-Edge-Cloud Hierarchical Federated Learning. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.
16. Mansour, Y.; Mohri, M.; Ro, J.; Suresh, A.T. Three Approaches for Personalization with Applications to Federated Learning. *arXiv* **2020**, arXiv:2002.10619.
17. Hao, M.; Li, H.; Luo, X.; Xu, G.; Yang, H.; Liu, S. Efficient and Privacy-Enhanced Federated Learning for Industrial Artificial Intelligence. *IEEE Trans. Ind. Inform.* **2020**, *16*, 6532–6542. [CrossRef]
18. Wang, H.; Kaplan, Z.; Niu, D.; Li, B. Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 1698–1707.
19. Guo, J.; Ho, I.W.-H.; Hou, Y.; Li, Z. FedPos: A Federated Transfer Learning Framework for CSI-Based Wi-Fi Indoor Posi-tioning. *IEEE Syst. J.* **2023**, *17*, 4579–4590. [CrossRef]
20. Karimireddy, S.P.; Kale, S.; Mohri, M.; Reddi, S.J.; Stich, S.U.; Suresh, A.T. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In Proceedings of the 37th International Conference on Machine Learning, PMLR, Online, 13–18 July 2020; Volume 119, pp. 5132–5143.
21. Tan, A.Z.; Yu, H.; Cui, L.; Yang, Q. Towards Personalized Federated Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *34*, 9587–9603. [CrossRef]
22. Xu, Y.; Fan, H. FedDK: Improving Cyclic Knowledge Distillation for Personalized Healthcare Federated Learning. *IEEE Access* **2023**, *11*, 72409–72417. [CrossRef]
23. Guo, S.; Xiang, B.; Chen, L.; Yang, H.; Yu, D. Multi-Level Federated Learning Mechanism with Reinforcement Learning Optimizing in Smart City. In *Proceedings of the Artificial Intelligence and Security*; Sun, X., Zhang, X., Xia, Z., Bertino, E., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 441–454.
24. Campolo, C.; Genovese, G.; Singh, G.; Molinaro, A. Scalable and Interoperable Edge-Based Federated Learning in IoT Contexts. *Comput. Netw.* **2023**, *223*, 109576. [CrossRef]
25. Liu, L.; Tian, Y.; Chakraborty, C.; Feng, J.; Pei, Q.; Zhen, L.; Yu, K. Multilevel Federated Learning-Based Intelligent Traffic Flow Forecasting for Transportation Network Management. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 1446–1458. [CrossRef]
26. Wu, Z.; Wu, X.; Long, Y. Multi-Level Federated Graph Learning and Self-Attention Based Personalized Wi-Fi Indoor Fingerprint Localization. *IEEE Commun. Lett.* **2022**, *26*, 1794–1798. [CrossRef]
27. Thapa, C.; Chamikara, M.A.P.; Camtepe, S.A. Advancements of Federated Learning Towards Privacy Preservation: From Federated Learning to Split Learning. In *Federated Learning Systems: Towards Next-Generation*; Rehman, M.H.U., Gaber, M.M., Eds.; Studies in Computational Intelligence; Springer International Publishing: Cham, Switzerland, 2021; pp. 79–109. ISBN 978-3-030-70604-3.
28. Ayad, A.; Renner, M.; Schmeink, A. Improving the Communication and Computation Efficiency of Split Learning for IoT Applications. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 01–06.
29. Thapa, C.; Chamikara, M.A.P.; Camtepe, S.; Sun, L. SplitFed: When Federated Learning Meets Split Learning. *Proc. AAAI Conf. Artif. Intell.* **2022**, *36*, 8485–8493. [CrossRef]
30. Tian, Y.; Wan, Y.; Lyu, L.; Yao, D.; Jin, H.; Sun, L. FedBERT: When Federated Learning Meets Pre-Training. *ACM Trans. Intell. Syst. Technol.* **2022**, *13*, 66:1–66:26. [CrossRef]
31. Jiang, H.; Liu, M.; Sun, S.; Wang, Y.; Guo, X. FedSyL: Computation-Efficient Federated Synergy Learning on Heterogeneous IoT Devices. In Proceedings of the 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS), Oslo, Norway, 10–12 June 2022; pp. 1–10.
32. Deng, R.; Du, X.; Lu, Z.; Duan, Q.; Huang, S.-C.; Wu, J. HSFL: Efficient and Privacy-Preserving Offloading for Split and Federated Learning in IoT Services. In Proceedings of the 2023 IEEE International Conference on Web Services (ICWS), Chicago, IL, USA, 2–8 July 2023; pp. 658–668.
33. Samikwa, E.; Maio, A.D.; Braun, T. ARES: Adaptive Resource-Aware Split Learning for Internet of Things. *Comput. Netw.* **2022**, *218*, 109380. [CrossRef]

34. Gao, Y.; Kim, M.; Abuadbba, S.; Kim, Y.; Thapa, C.; Kim, K.; Camtep, S.A.; Kim, H.; Nepal, S. End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things. In Proceedings of the 2020 International Symposium on Reliable Distributed Systems (SRDS), Shanghai, China, 21–24 September 2020; pp. 91–100.
35. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
36. Shahri, E.; Pedreiras, P.; Almeida, L. Extending MQTT with Real-Time Communication Services Based on SDN. *Sensors* **2022**, *22*, 3162. [CrossRef] [PubMed]
37. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
38. Tschandl, P.; Rosendahl, C.; Kittler, H. The HAM10000 Dataset, a Large Collection of Multi-Source Dermatoscopic Images of Common Pigmented Skin Lesions. *Sci. Data* **2018**, *5*, 180161. [CrossRef] [PubMed]