



北京邮电大学
Beijing University of Posts and Telecommunications



Queen Mary
University of London

Undergraduate Project Report 2020/21

Federated Learning Platform Design for Edge Computing

Name: *****
School: International School
Class: *****
QM Student No.: *****
BUPT Student No.: *****
Programme: e-Commerce
Engineering with Law

Date: 21-04-2021

Table of Contents

Abstract.....	3
Chapter 1: Introduction	5
Chapter 2: Background.....	8
2.1 Federated Learning	8
2.1.1 Centralized Machine Learning Models	8
2.1.2 Stochastic Gradient Descent	9
2.1.3 Federated SGD	10
2.1.4 Federated Averaging	10
2.1.5 HierFAVG	11
2.2 Edge Computing.....	12
2.2.1 Concept and Features of Edge Computing	13
2.3 Background of Tools and Frameworks.....	14
2.3.1 Edge Computing Platform: KubeEdge	14
2.3.2 Federated Learning Framework: Flower	16
2.3.3 Raspberry pi.....	16
Chapter 3: Design and Implementation	17
3.1 Federated Learning Model Design	17
3.1.1 The Simplest Model: Model-1	17
3.1.2 Cloud-Edge Collaboration: Model-2	18
3.1.3 HireFAVG: Model-3	19
3.2 Edge Computing Architecture Design	20
3.3 Deployment of KubeEdge in China.....	20
3.3.1 Deployment of CloudCore	21
3.3.2 Deployment of EdgeCore.....	22
3.4 Implementation of Cloud Management System	22
3.4.1 configProcessor	23
3.4.2 yamlProcessor.....	23
3.4.3 K8sClient	23
3.4.4 dynamicClient	24
3.4.5 crdClient.....	24
3.4.6 Controller.....	25
3.4.7 Template	27
3.4.8 Services	27
3.5 Implementation of Federated Learning.....	27
3.5.1 Client.....	28
3.5.2 Learner.....	30
3.5.3 FL_Driver.....	31
3.5.4 Custom ML Model	31
3.5.5 MQTT Client.....	31
3.6 Implementations of Different Models.....	31
3.6.1 Model-1 and Model-2.....	32
3.6.2 Model-3	32
3.7 Security and Certification	33
3.7.1 SSL/TLS.....	33
3.7.2 Device authentication.....	34
Chapter 4: Result and Discussion	35

Federated Learning Platform Design for Edge Computing

4.1.1 Devices Information.....	35
4.1.2 Result of Model-1 and Model2.....	35
4.1.3 Result of Model-3	37
4.1.4 Reliability Test Result	38
Chapter 5: Conclusion and Further Work.....	40
References	42
Acknowledgement	44
Risk and environmental impact assessment.....	45

Abstract

With the development of mobile Internet and its applications, a large amount of data is generated every day by various devices from all over the world. Combined with the continuous rise of computer computing power in the past decade, deep learning has become a hot topic. However, a large amount of data needs to be uploaded to the central server for deep learning, which not only consumes a large amount of bandwidth and storage resources, but also may lead to the disclosure of users' privacy. Therefore, some scholars have proposed federated learning. In federated learning, users can first conduct machine learning training locally, and then upload the training parameters to the server for aggregation. However, in some cases, users themselves do not have enough computing capacity to train, and the model parameters will still occupy a lot of network resources. At this point, edge computing can solve some of the defects of federated learning, and achieve higher efficiency. This project builds a federated learning platform based on KubeEdge. Through the cooperation of cloud, edge and clients, the system can be compatible with a variety of learning models and realize efficient machine learning. In addition, the scheduling capabilities and the interface provided by the design facilitate secondary development. Experiments show that the federated learning system has little difference from the traditional federated learning system, and has a certain fault-tolerant ability.

摘要

随着移动互联网及其应用的发展，来自世界各地的各类终端每天会产生大量数据。结合近十年内计算机算力的不断上升，深度学习成为了一个热点话题。然而，大量的数据需要上传到中心服务器才能进行深度学习，这不但需要消耗大量的带宽与存储资源，还可能造成用户的隐私泄露。因此，一些学者提出了联邦学习。在联邦学习中，用户可以先在本地进行机器学习训练，再将训练的参数上传至服务器进行聚合。然而，一些情况下，用户本身并没有足够的计算能力进行训练，且模型的参数仍然会占用不少网络资源。这时候，采用边缘计算可以解决联邦学习的一部分缺陷，实现更优秀的性能、更低的成本与更高的效率。本项目基于 KubeEdge 搭建了一套基于边缘计算的联邦学习框架，通过云-边-端的共同协作，系统可以兼容多种不同的联邦学习模式，并实现高效率的机器学习。此外，KubeEdge 提供的调度能力与设计本身提供的接口有利于开发者进行二次开发。实验表明在该系统上运行的联邦学习与传统的联邦学习性能差别较小，且具有一定的容错能力。

Chapter 1: Introduction

With the development of mobile Internet and high-performance mobile terminals, mobile terminal users can easily access the Internet by using cellular mobile data or WLAN, so as to enjoy the convenience brought by various applications (LUO, WU and YANG, 2011). In the use of users, extensive and rich applications are producing lots of data every day. The huge amount of data brings great value and opportunities, and many enterprises and research institutions are eager to make use of this data. In recent years, the emergence of deep neural network technology, can make full use of this data for machine learning, to achieve some amazing systems and create unimaginable values (Naimi and Westreich, 2014). At the same time, the huge amount of data also poses many challenges to machine learning. In many cases, the data is distributed, and the collection of data often consumes a lot of computing and communication resources. In addition, some data security issues related to personal privacy are also worthy of attention. These are all things that engineers need to think about in the context of big data and machine learning.

To solve the problem of data security and data resource integration in machine learning, some scholars proposed federated learning, which is a novel machine learning framework. Through Federated Learning, different users and organizations can conduct machine learning that meets the requirements of privacy, data security and government laws (Yang, Liu, Chen and Tong, 2019). Specifically, Federated learning is divided into two steps: learning and aggregation. In the learning process, each user will use their own data for machine learning, and get a preliminary training model. In the aggregation process, users need to upload their preliminary models to a cloud server, and the server will aggregate the models using some specific strategies. Through these two steps, enterprises or organizations can achieve machine learning training without obtaining specific user data. It is very effective to protect the user's data, and avoid large-scale data transmission. In addition, Federated Learning can also be used to aggregate data between two enterprises, generating machine learning models without exchanging confidential data. To sum up, federated learning is a machine learning framework in the distributed environment, which may play a significant role in the future intelligent medical care, intelligent finance and other systems involving user privacy and large amounts of data.

However, Federal Learning also has some obvious problems (Kairouz and McMahan, 2021). During the learning process, the data is trained locally, which requires the terminal device to have enough computing resources. Today's common smart phones may have some computing

power, but they are nowhere near as powerful as GPUs designed to train deep learning models. In addition, the terminal devices may consume a lot of energy and generate much heat when training the machine learning model, which is unacceptable to some users. During the aggregation process, the user needs to upload models to the cloud server. When the number of users is quite large, the server might be stressed. Also, the uploading of models will consume communication and network resources, or even cause network congestion. To solve these problems, we can adopt a novel network architecture: edge computing.

Edge computing refers to an open platform that integrates network, computing, storage and application capabilities on the side close to the data source to provide services nearby. Terminal devices can initiate requests locally and the edge nodes can respond quickly, thereby reducing network delays and resource consumption, or helping devices acquire more computing resources (Shi et al., 2016). Therefore, if federated learning is running in the environment of edge computing, some problems caused by federated learning can be avoided through the overall design of the system. For example, a lot of network resources can be saved if the device only uploads the model to the edge server, where it is aggregated. If the device uploads data to a trusted edge node and then machine learning is carried out by a high-performance edge server, the problem of insufficient computing resources of the device can be avoided. In addition, because each edge server can only cover a subset of users, efficient federated learning requires a well-designed system architecture. At present, the common open-source edge computing platforms are KubeEdge and K3S. Among them, KubeEdge is more suitable to be combined with federated learning (Because KubeEdge is lighter), so we will use KubeEdge to build the edge computing platform.

This project focuses on the combination of edge computing and federated learning, and gives full play to the advantages of edge computing to implement a secure, reliable, efficient and usable federated learning platform. Specifically, the project is divided into two parts: design and implementation.

In the design part, we propose three different federated learning architectures in the context of edge computing based on the existing edge computing theories and federated learning strategies. The three models are called Model-1, Model-2, and Model-3, respectively. In Model-1, the user performs machine learning locally, which is then aggregated by the edge server. In Model-2, the user uploads the data to the edge, and the edge server participates in the machine learning process and is aggregated by the cloud server. Model-3 is a quiet complex client-edge-cloud collaboration model. These three models have their own advantages and disadvantages, which

will be introduced later.

In the implementation part, we first use KubeEdge to build an edge computing platform. This platform can realize the access, maintenance, management and application deployment of edge devices. At the same time, because KubeEdge is based on Kubernetes, it naturally provides the ability of edge cloud collaboration. On this platform, we use Python language, Flower framework and PyTorch library to achieve three federated learning models. We also discuss the security issues of edge computing and federated learning in the concrete implementation. In addition to the management signal encryption provided by KubeEdge native, we use SSL/TLS to encrypt the data throughout the transmission to achieve a secure edge network. After the programming was completed, we used several Raspberry Pi 3B+ and Linux virtual machine to conduct the simulation. The results show that the platform can run normally and stably, and has a certain robustness. Combined with the simulation results, the three models can achieve efficient federated learning from the perspective of machine learning. The performance of model-3 is very similar with that of the centralized machine learning model.

The rest of this report is organized as follows. In Chapter 2, we will introduce the background to federated learning and edge computing, as well as the features and details of the tools and frameworks used in the implementation. In Chapter 3, we will show that the implementation of entire platform, includes sever-side and client-side. In Chapter 4, the simulation results are provided, showing that the system has good performance. The conclusion and further work are given in Chapter 5.

Chapter 2: Background

In this chapter, we will first introduce the mathematical model and operation mechanism of federated learning. Second, we will summarize the history, design philosophy and operation of edge computing. Finally, the relevant tools and frameworks that have appeared in this project will be described in detail.

2.1 Federated Learning

In most industries, data exists in the form of islands. For users, a large amount of data is distributed on the terminals of different users, which is affected by laws and regulations and cannot be uploaded to the central server. For enterprises, due to industry competition, privacy security, complex administrative procedures and other issues, even the realization of data integration between different departments of the same company is also faced with many obstacles. It is almost impossible to integrate the data scattered in various places and institutions. In order to meet the requirements of data privacy, security and supervision, so that artificial intelligence systems can more efficiently and accurately use their own data together and solve the problem of data isolation, some scholars proposed Federated Learning.

Federated learning is a machine learning framework that effectively enables multiple organizations to use data and conduct machine learning in a manner that meets user privacy, data security, and other regulations. In the White Paper on Federated Learning, Federated learning is defined as follows: In the process of machine learning, each participant can use the data of other parties to conduct joint modelling. There is no need for parties to share data resources. In other words, data joint training is conducted to establish a shared machine learning model under the condition that the data is not out of the local area.

By definition, federated learning is a special kind of machine learning. Next, we will derive the mathematical model of federated learning from a basic machine learning perspective and introduce some aggregation strategies.

2.1.1 Centralized Machine Learning Models

Before introduce federated learning, we need to model traditional centralized machine learning. First of all, all machine learning problems with non-convex loss functions, such as neural networks, can be modelled by the following formula:

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w) \quad (1)$$

In the above formula, $f(w)$ is the objective function we need to optimize, w is the parameter of the model, and $f_i(w)$ refers to the loss caused by the i -th parameter. That is, for machine learning problems, we need to adjust the parameters so that the sum of the losses generated by each parameter is minimized.

The above formula is too general, we need to refine it further. We can assume that:

$$f_i(w) = \ell(x_i, y_i; w) \quad (2)$$

This formula means that (x_i, y_i) contributes to loss $f_i(w)$ in the parameter w , which is the same as the original formula. If we have K users contributing the training data sets, and the data set contributed by each user is \mathcal{P}_k . The number of elements in each set is $n_k = |\mathcal{P}_k|$, then we can rewrite the above formula as:

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w) \quad (3)$$

From the above derivation and introduction, we can find that the traditional centralized machine learning model can be split to a decentralized model.

2.1.2 Stochastic Gradient Descent

Most machine learning or deep learning algorithms involve some form of optimization. In general, deep learning is trained using gradient descent. Gradient Descent is an iterative algorithm, whose principle is to calculate the gradient of the objective function repeatedly, and then adjust the parameters along the gradient direction, so as to get the local minimum value.

In gradient descent, there are many different strategies for finding gradients. Batch Gradient Descent (BGD) is a simple strategy, which uses all data to update the gradient in each iteration. In this strategy, the orientation determined by the whole data set can better represent the sample population, thus more accurately pointing in the direction of the extreme value. When the objective function is convex, it will converge to the global minimum, and if the objective function is not convex, it will converge to the local minimum.

Stochastic Gradient Descent (SGD) is different from Batch Gradient Descent in that only one sample is used to update the parameters during each iteration. This approach leads to a very blind algorithm with a large number of iterations, not necessarily convergent, and therefore is not commonly used. Mini-batch Gradient Descent (MBGD) falls somewhere in between, using more than one but not all training samples. In this practice, the iterations are fast because only part of the data set is selected at a time. In addition, using a part of data each time can greatly reduce the number of iterations required for convergence and ensure the convergence of the

algorithm at the same time. Mini-batch Gradient Descent is commonly referred to as stochastic gradient descent, as we will do later in this article.

2.1.3 Federated SGD

In federated learning, we can think of data from each user as a batch of data. Then, each batch of data was trained separately (McMahan et al., 2021). This idea could be called Federated SGD (FedSGD).

FedSGD is a very basic Federated Learning strategy. In each round of machine learning, all users need to participate in the training with all the local data, and then upload to the server for aggregation. When aggregating, the server needs to follow the following principles:

First, we need to have a learning rate η . Then, for any user k , the current gradient needs to be calculated, which is:

$$g_k = \nabla F_k(\omega_t) \quad (4)$$

Where, w_t represents the model parameters of the user at this time. Then the server needs to collect the gradient of each user and update the model with the mean value of g_k .

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^K \frac{n_k}{n} g_k \quad (5)$$

From the derivation in the previous section, $f(w)$ can also be expressed as follows:

$$\sum_{k=1}^K \frac{n_k}{n} g_k = \nabla f(\omega) \quad (6)$$

Therefore, the policy for aggregation can also be written as:

$$\forall k, \quad \omega_{t+1}^k \leftarrow \omega_t + \eta g_k \quad (7)$$

Combining the above formulas, we can get:

$$\omega_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} \omega_{t+1}^k \quad (8)$$

The above formula clearly expresses the principle of FedSGD. The core of this is to average the models from each user to get a new model. In addition, each user can iterate several rounds locally before aggregation:

$$\omega^k \leftarrow \omega^k + \eta \nabla F_k(\omega_k) \quad (9)$$

This concludes our introduction to the principles of FedSGD. Next we'll look at a more common practice, FedAvg.

2.1.4 Federated Averaging

Federated Averaging (FedAvg) is a federated learning strategy for deep learning proposed by

McMahan. This strategy is simple to implement and is widely used by various federated learning frameworks. In fact, FedAvg is a special case of FedSGD, and the specific strategies are as follows:

We need to define three parameters first:

C : The proportion of the total number of clients participating in the federated learning aggregation per round.

E : The number of iterations each client has trained locally.

B : Batch size of each client in local training.

Then, the FedAvg process is shown below:

```

Server executes:
initialize  $w_0$ 
for each round  $t = 1, 2, \dots$  do
   $m \leftarrow \max(C \cdot K, 1)$ 
   $S_t \leftarrow$  (random set of  $m$  clients)
  for each client  $k \in S_t$  in parallel do
     $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ 
   $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ 

ClientUpdate( $k, w$ ): // Run on client  $k$ 
 $\mathcal{B} \leftarrow$  (split  $\mathcal{P}_k$  into batches of size  $B$ )
for each local epoch  $i$  from 1 to  $E$  do
  for batch  $b \in \mathcal{B}$  do
     $w \leftarrow w - \eta \nabla \ell(w; b)$ 
return  $w$  to server
    
```

Figure 1 Pseudo code of FedAvg

It can be seen that when $C=E=1$ and B is infinite, FedAvg is the same as FedSGD.

2.1.5 HierFAVG

The two federated learning strategies described earlier are classic and work well in traditional cloud server frameworks. However, FedAvg has some inherent defects for the client-edge-cloud collaboration model. Specifically, the edge server cannot cover all the clients. If we have multiple edge servers, Federated Learning may generate multiple different models.

HierFAVG was created to address the deficiency of federated learning in edge computing (Liu, Zhang, Song and Letaief, 2020). In this model, the user's data is trained locally and then aggregated by the edge server. After the edge servers have been aggregated multiple times, the models will be uploaded to the cloud server and aggregated again. In addition, this twice-aggregated structure can be stacked multiple times to form a hierarchy. As shown in the figure below:



Figure 2 Process of HierFAVG

In the implementation, we need to adjust the number of edge aggregation to achieve the best performance. Liu's paper shows that this structure converges in a variety of situations, and the effect is similar to that of ordinary Federated Learning.

2.2 Edge Computing

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the location where it is needed to improve response times and save bandwidth. With edge computing, we can achieve many complex applications. For example, video sites can set up video storage servers at the edge, and users only need to pay very low bandwidth costs to enjoy high quality online video. Factories can set up large-scale sensor networks using edge computing. Through the powerful data processing feature of the edge server, the factory can detect the production status in real time. Autonomous vehicles also need edge computing support. Edge servers can provide road information and high-precision maps for autonomous vehicle. In addition, edge computing has relevant applications in smart medicine, smart finance and machine learning.

The figure below shows the application of edge computing in smart cities (Liu et al., 2020). The edge nodes are deployed beside the base station and connect with the cloud server through the core network. A wide variety of applications can be connected to edge nodes to improve service quality.

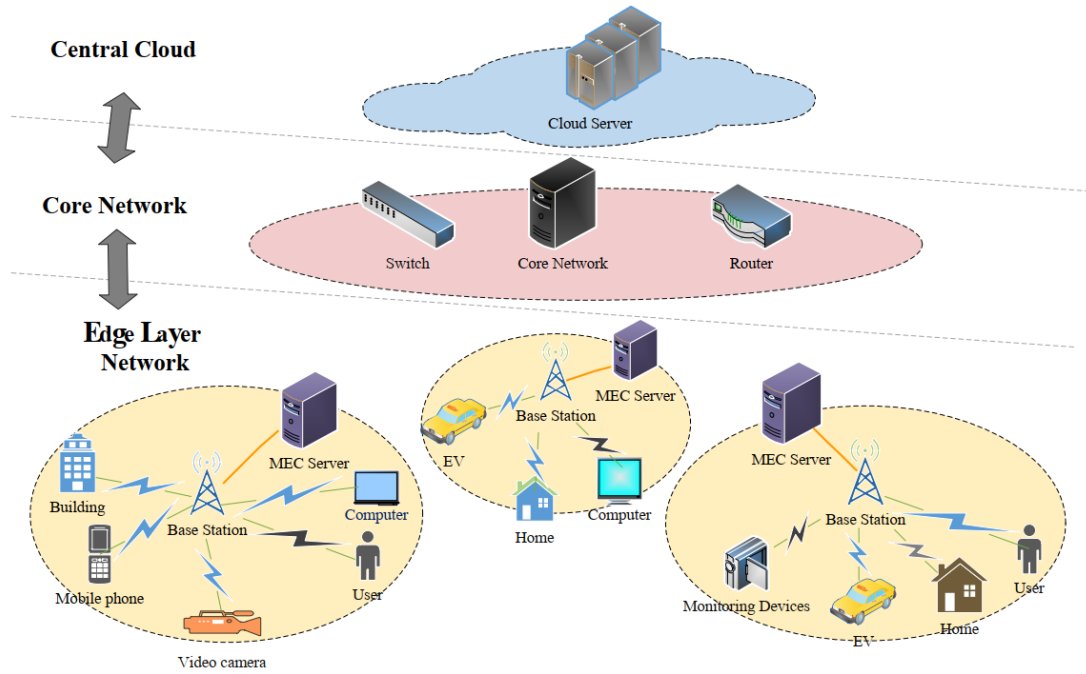


Figure 3 Application of edge computing

2.2.1 Concept and Features of Edge Computing

As for edge computing, its shape, location and performance are different from cloud servers, so this emerging distributed system design scheme has its own characteristics in concept and features (Shi and Dustdar, 2016).

First, because edge servers are deployed on the edge, their security and privacy issues need to be re-examined. On the one hand, because part of the user data is stored in the edge of the small network, it is born with certain security characteristics. The end user can have a great deal of autonomy over the data at the edge. On the other hand, edge devices may be limited by resources, so need to choose a specific encryption method. Security strategies in cloud-edge collaboration are also important.

Second, Edge Computing may involve a large number of device connections, where system reliability needs to be considered. For edge-side devices, when the edge node fails, they should still be able to provide all or part of the service until the edge node is brought back online. For the edge node, it must provide the ability of edge autonomy, such as device management and service maintenance. In a cloud-edge collaborative system, when the cloud service fails, the edge nodes need to be able to withstand the pressure of the whole system.

Third, edge system should also be flexible. On the edge side, due to the heterogeneity of the system and the shortage of edge resources, the scalability of the system will be limited at some cases. Security should also be considered when edge nodes are scaled up and expanded.

Finally, the edge computing framework itself should be extremely efficient and low-cost. If edge computing is more expensive and slower than cloud computing, there is no need for edge computing at all.

2.3 Background of Tools and Frameworks

In this part, all of the tools and frameworks that appear in this project are described.

2.3.1 Edge Computing Platform: KubeEdge

In order to realize device management, connection and network configuration in edge computing, a set of complete edge computing framework is necessary. Implementing a complete framework from scratch takes a lot of time and resources. The emergence of KubeEdge has helped the industry solve this difficulty.

KubeEdge is an open source edge computing framework built on Kubernetes that provides core infrastructure support for network, application deployment, and metadata synchronization between the cloud and Edge, enabling containerized applications to be deployed and managed on the edge (Xiong, Sun, Xing and Huang, 2018). KubeEdge also supports MQTT and allows developers to extend more connections to help edge devices communicate with edge nodes on limited network resources. Specifically, KubeEdge consists of a cloud part and an edge part. As shown in the figure below:

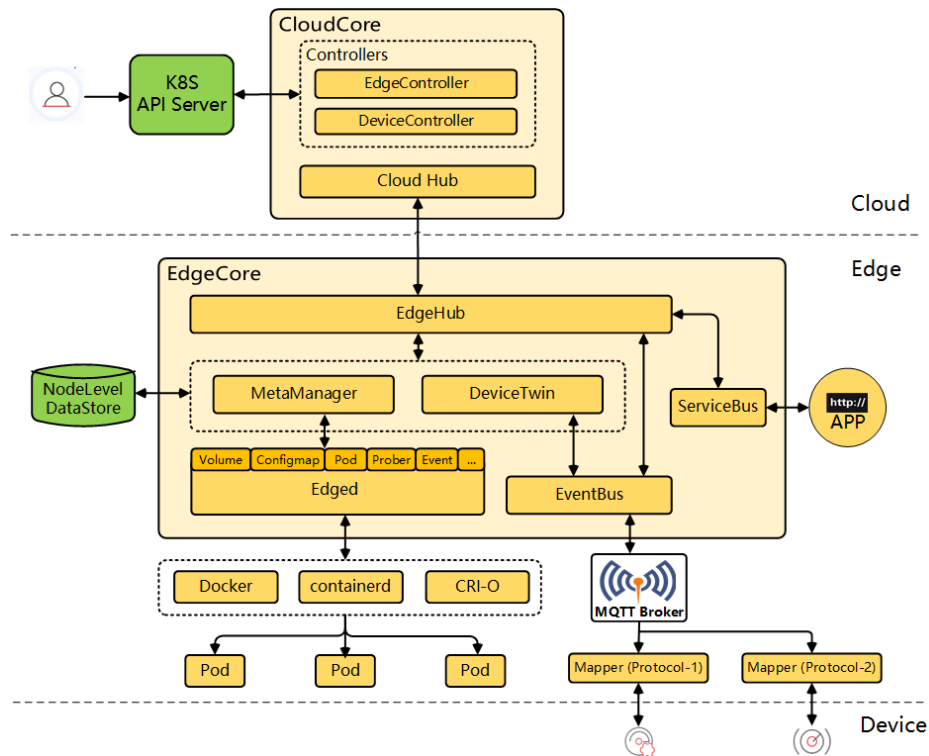


Figure 4 The structure of KubeEdge

CloudCore

In the cloud, the main component of KubeEdge is CloudCore. In fact, CloudCore is a plug-in of Kubernetes, which interacts with Kubernetes in a non-intrusive way. CloudCore has three important components, EdgeController and DeviceController, as well as a communication module, CloudHub. User's operations on KubeEdge (or Kubernetes) will be sent to EdgeController by an API server, which will process them and then send them to CloudHub, then, send to edge. The DeviceController is responsible for modelling the edge devices in the form of Kubernetes CRDS and synchronizing the information of the edge devices.

EdgeCore

EdgeCore can be seen as a very lightweight K8S with a low resource footprint and can therefore be deployed on a variety of edge devices. First, EdgeCore allows for the self-management of edge devices. For example, when the state of the edge device changes, Edgecore actively uploads the change to the cloud. Secondly, EdgeCore has implemented the maintenance of services in the edge node containers. When the cloud needs to deploy applications to the edge, EdgeCore can take the initiative to schedule, complete container creation, image pulling and other operations.

In EdgeCore, edge devices access EventBus via the MQTT protocol, or by using other protocols transform to the MQTT protocol by the Mapper. EventBus can forward messages to DevicetTwin to synchronize the cloud side state, or send them directly to EdgeHub and upload them to the cloud directly. The scenario of downloading from the cloud is similar to uploading.

Advantages and Disadvantages of KubeEdge

As an emerging open source project, KubeEdge has outstanding advantages. First of all, since the CloudCore is connected directly to the K8S, it is compatible with various K8S APIs, so new users can get started quickly. Secondly, KubeEdge provides a complete cloud-edge collaboration system, which realizes the autonomy of the edge system. In addition, KubeEdge inherits the K8S concept and brings the cloud-native model to the edge.

As KubeEdge is still in development, its drawbacks are numerous. For example, KubeEdge's encryption and security scheme is very simple, and it does not provide enough technical support to guarantee the security of devices in the edge network. In addition, the overall framework is not tightly connected to the edge devices and only maintains a weak connection through MQTT or some custom protocols. However, the framework uses K8S CRDS to maintains device instances, and this heavier approach is not a good match for weak connections.

2.3.2 Federated Learning Framework: Flower

Flower is an open-source federated learning framework written in Python that provides a federated learning interface for multiple platforms and approaches. Specifically, Flower supports mainstream deep learning platforms such as PyTorch and TensorFlow. Its server uses gRPC for network connection, and can customize the learning strategy. However, because of the simplicity of this framework, there is no support for data and privacy protection. In sum, using the Flower framework for Federated Learning allows developers to focus on the implementation of the algorithm without having to worry about other details such as network connections.

2.3.3 Raspberry pi

Raspberry Pi is a microcomputer based on an ARM core. Since its introduction, it has been embraced by computer enthusiasts and geeks alike. Raspberry Pi has all the basic functions of a PC, which is very suitable for project development and simulation testing. In this project, we use Raspberry Pi 3B+ as the edge node.

Chapter 3: Design and Implementation

In order to realize federated learning in edge computing, give full play to the advantages of edge computing and improve the performance of federated learning, we need to design and implement this system carefully. In this section, three Federated Learning models in edge computing systems will be introduced. Later, the network topology and management mode will be described in detail. Finally, we'll explain the implementation of the three different models in detail, including some code analysis.

3.1 Federated Learning Model Design

In this project, many users come together to provide the training model of their own data and the training process will be done by a reliable device. To implement FL, we need to determine where the data is stored and where the model is aggregated. So we designed three models.

3.1.1 The Simplest Model: Model-1

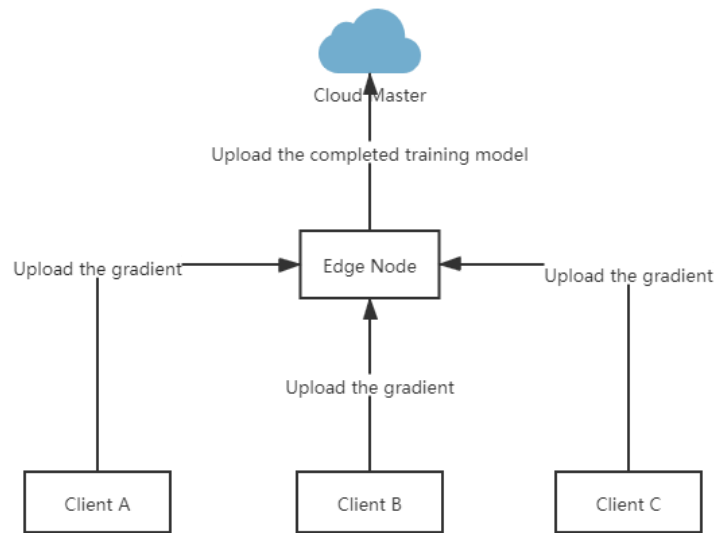


Figure 5 Model-1

The figure above is the most traditional federated learning architecture. In this scheme, the client needs to process the data by itself, upload the encrypted gradient to the edge node, and then the edge server aggregates the data to generate the trained model. The benefits of such a scheme are obvious. The client does not need to upload sensitive data, and the overall system is very simple.

However, Model-1 is more of an experimental application than a practical one. Because this model does not bring all the advantages of edge computing into play, it will bring some

problems. First of all, one edge node cannot cover all the devices in every corner of the world, so we can only get a small amount of data, which leads to performance degradation of the machine learning model. Secondly, this model requires the client to have very strong computing power. For some mobile devices, this condition is unacceptable. Nonetheless, Model-1 also has a purpose. This simple, easy to deploy architecture is ideal for testing system stability. If Model-1 can run successfully, then other more complex structures should be able to run stably in edge computing systems as well.

3.1.2 Cloud-Edge Collaboration: Model-2

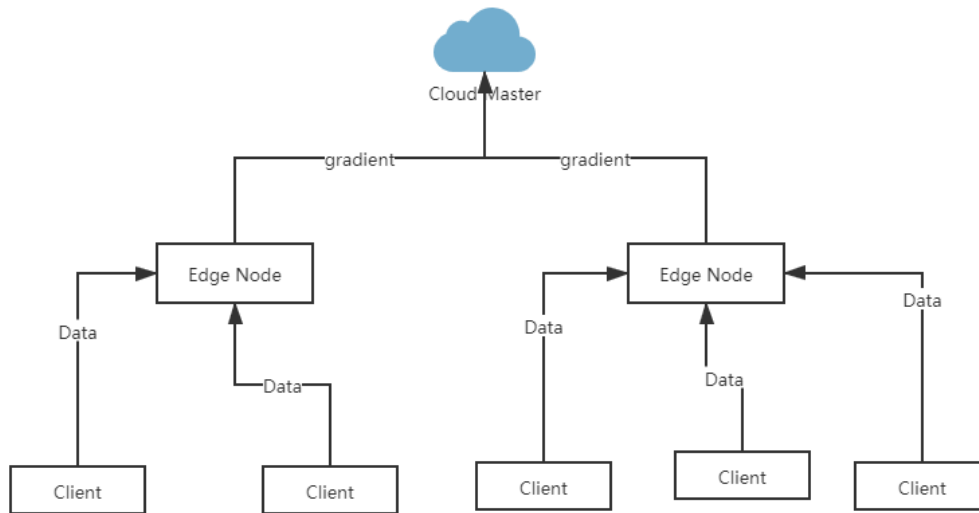


Figure 6 model-2

The figure above is an edge computing strategy of edge cloud collaboration. In this model, we assume that the edge node is trusted, so the client can safely transfer data to the edge node. The gradient is then calculated by the edge node and uploaded to the cloud server for aggregation. The final model will be generated in the cloud. This model relies heavily on the computing power of the edge nodes, where machine learning is all done. In addition, the reliability of the connection between cloud server and edge server also needs to be guaranteed.

Model-2 is a more practical model, because the client does not need to pay resources for machine learning because of the existence of edge nodes. In terms of data privacy, the isolation of edge network itself can also ensure the security of user data. However, due to the need to synchronize a large amount of data in the side cloud collaboration, the security in the data exchange process needs to be carefully considered.

3.1.3 HireFAVG: Model-3

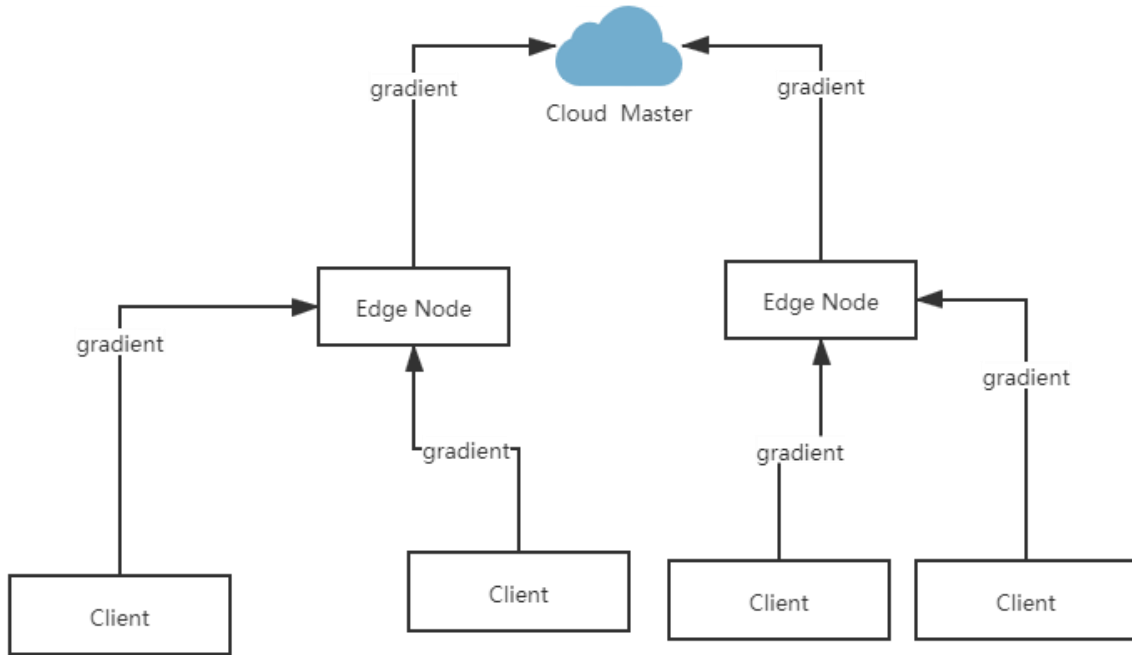


Figure 7 model-3

The figure above shows the HierFAVG model mentioned earlier. We're going to call it Model-3 in this case. In the case of edge computing, we think this model is the most useful federated learning model with the most potential. In Model-3, client will be trained k_1 times and then aggregated by the edge server. When the edge server aggregates k_2 times, it is uploaded to the cloud server for aggregation once. In this way, different data from different regions and different edge networks can be aggregated into the cloud server, which helps the system to generate better performance models. In addition, the amount of data uploaded to the cloud by the edge node is significantly smaller than that uploaded to the edge node by the client, which helps to save network communication resources.

3.2 Edge Computing Architecture Design

As the basis of the system, the network and computation model of edge computing should also be considered. In order to work with KubeEdge, the edge computing architecture must be designed as a cloud-edge collaboration pattern. This is shown below:

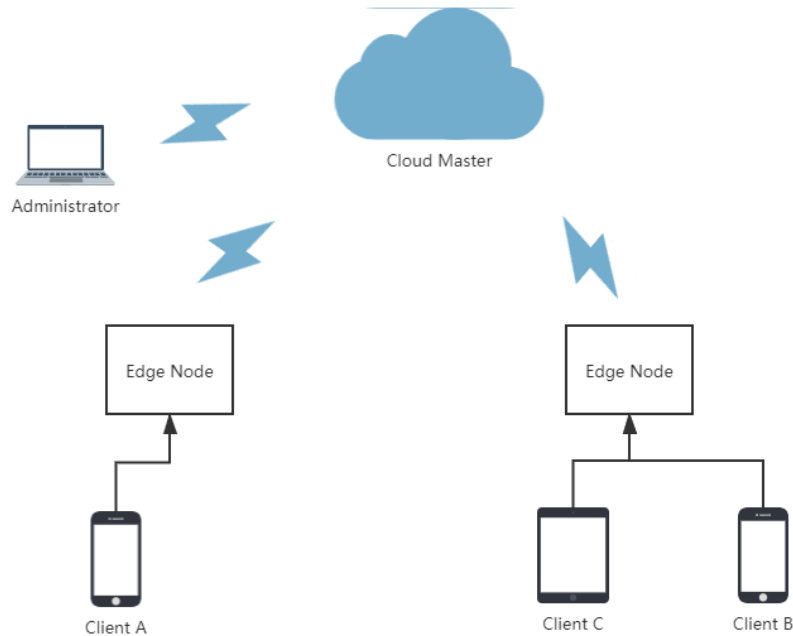


Figure 8 3.2 Edge computing architecture

As can be seen from the figure above, the core of the whole system is the Cloud Master. The cloud server should be able to be controlled by an external administrator and have the highest authority for the entire system. The edge nodes are set in the same network as the edge devices to minimize latency. In addition, Edge nodes have a strong autonomous ability. When the cloud server is offline, the edge server can provide service maintenance and edge network governance ability. The administrators can also access the edge server in a certain way. Also, the whole system is heterogeneous, and the edge node needs to allow access to many different types of devices.

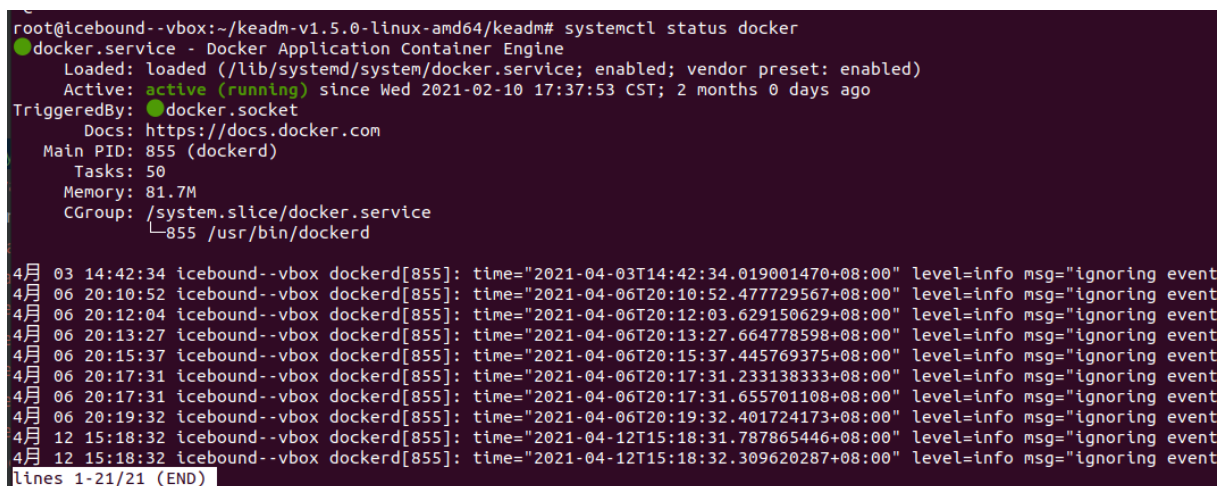
3.3 Deployment of KubeEdge in China

After the system design, in order to implement the system, we first need to deploy KubeEdge to build the edge computing platform. In China, for a variety of reasons, the deployment of KubeEdge is plagued by network problems and requires many special methods to be tried. In addition, KubeEdge deployment on Raspberry Pi presents unique challenges.

3.3.1 Deployment of CloudCore

First, in order to install CloudCore, we need a deployed Kubernetes node and a container capable of running Kubernetes applications.

First, we need to install Docker to meet the requirements of Kubernetes. Because the KubeEdge project shows up to Kubernetes 1.19, the corresponding Docker can't be the latest version either. Then, we need to use apt or yum to install a specific version of Docker. In this process, due to network environment problems, we need to use some Chinese mirror images of the software source.



```

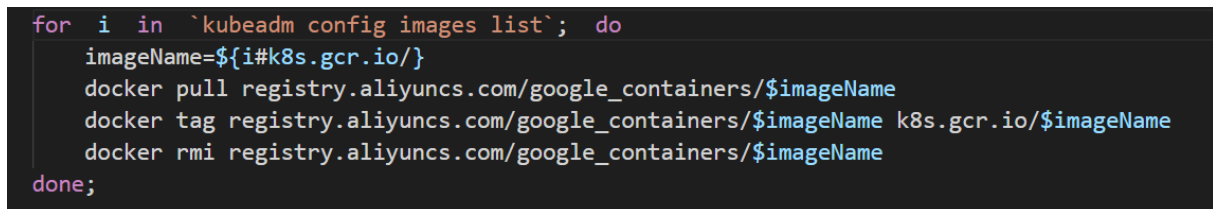
root@icebound--vbox:~/keadm-v1.5.0-linux-amd64/keadm# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-02-10 17:37:53 CST; 2 months 0 days ago
     TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
      Main PID: 855 (dockerd)
        Tasks: 50
       Memory: 81.7M
      CGroup: /system.slice/docker.service
              └─855 /usr/bin/dockerd

4月 03 14:42:34 icebound--vbox dockerd[855]: time="2021-04-03T14:42:34.019001470+08:00" level=info msg="ignoring event
4月 06 20:10:52 icebound--vbox dockerd[855]: time="2021-04-06T20:10:52.477729567+08:00" level=info msg="ignoring event
4月 06 20:12:04 icebound--vbox dockerd[855]: time="2021-04-06T20:12:03.629150629+08:00" level=info msg="ignoring event
4月 06 20:13:27 icebound--vbox dockerd[855]: time="2021-04-06T20:13:27.664778598+08:00" level=info msg="ignoring event
4月 06 20:15:37 icebound--vbox dockerd[855]: time="2021-04-06T20:15:37.445769375+08:00" level=info msg="ignoring event
4月 06 20:17:31 icebound--vbox dockerd[855]: time="2021-04-06T20:17:31.233138333+08:00" level=info msg="ignoring event
4月 06 20:17:31 icebound--vbox dockerd[855]: time="2021-04-06T20:17:31.655701108+08:00" level=info msg="ignoring event
4月 06 20:19:32 icebound--vbox dockerd[855]: time="2021-04-06T20:19:32.401724173+08:00" level=info msg="ignoring event
4月 12 15:18:32 icebound--vbox dockerd[855]: time="2021-04-12T15:18:31.787865446+08:00" level=info msg="ignoring event
lines 1-21/21 (END)

```

Figure 9 The status of docker

Secondly, we need to deploy the K8s. Like Docker, it can be installed via APT. The specific installation instructions are: `apt-get install -y kubelet=1.19.7-00 kubeadm=1.19.7-00 kubectl=1.19.7-00`. After that, we also need to pull out the core image of the K8S with the following script:



```

for i in `kubeadm config images list`; do
    imageName=${i#k8s.gcr.io/}
    docker pull registry.aliyuncs.com/google_containers/$imageName
    docker tag registry.aliyuncs.com/google_containers/$imageName k8s.gcr.io/$imageName
    docker rmi registry.aliyuncs.com/google_containers/$imageName
done;

```

Figure 10 Script to get K8S base image in China

Finally, execute docker images to check that the images are ready. If they are, initialize the node with the following command: `Kubeadm init --pod-network-cidr=10.244.0.0/16`

Then, execute `kubectl get nodes -o wide` to see if the node is ready. If the master node is ready, then K8S has been installed successfully.

After the K8S is ready, we can deploy KubeEdge using the officially provided kdeam tool. Just

execute `./keadm init` and then the CloudCore is ready.

3.3.2 Deployment of EdgeCore

The EdgeCore does not need K8S, but Docker. Docker version had better avoid problems just like Cloud. First we need to get a token on the cloud server using `./keadm gettoken`. Then execute `./keadm join` at the edge to join the system.

Finally, go to the cloud server and execute: `kubectl get node`. We can see the nodes, indicating that they are connected.

3.4 Implementation of Cloud Management System

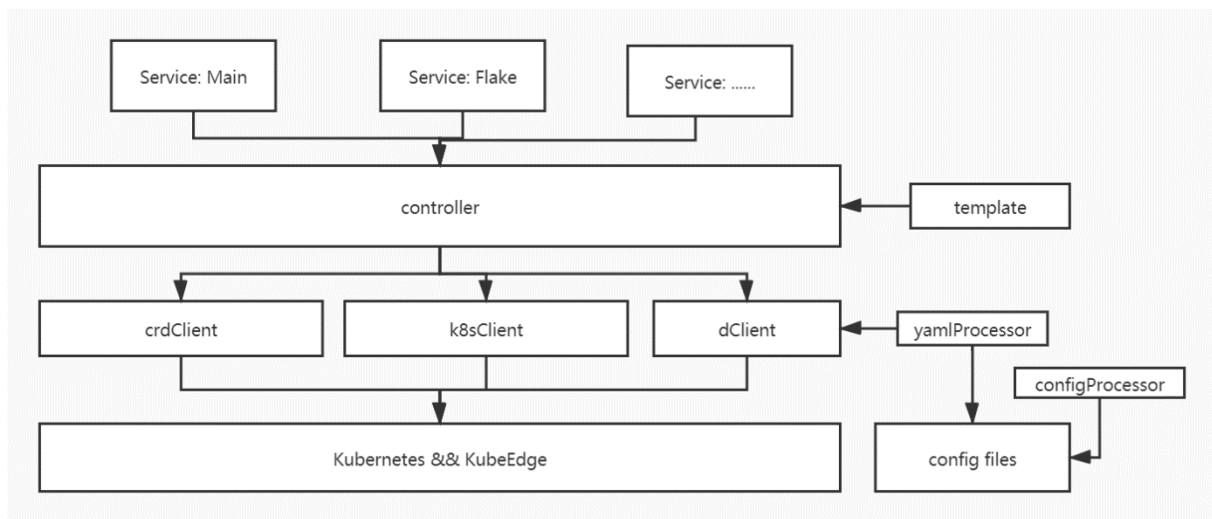


Figure 11 The structure of Cloud Management System

In order to realize the system's control over KubeEdge, we designed a modular and extensible cloud management system. The underlying system consists of a variety of Go-Client components. The `crdClient` component is used to manage resources, the `k8sClient` is used to get various information from kubernetes. And the `dclient` is a flexible and dynamic component for directly manipulating YAML configuration files. In addition, a configuration component can read the system's configuration from the file system. In the middle of the system is the controller, which is responsible for disassembling various complex tasks into a combination of actions of different underlying components, and controlling these components to perform tasks as required. At the top of the system is a set of extensible services. Because controller opens up various interfaces, developers can develop various services in a low-coupling manner. For example, we developed a Main service to demonstrate different federated learning models, and an HTTP server to allow external devices to actively access the system. In sum, the system has the basic functions, but also provides the ability to modify and upgrade later.

3.4.1 configProcessor

The ConfigProcessor module is implemented in config.go and mainly consists of two methods: OutClusterConf and InClusterConf. These two methods are used to retrieve environment information from outside or inside the cluster and are integrated into a rest.config object. Specifically, OutClusterConf should be used if the controller's upper-level services are running outside of the KubeEdge cluster, otherwise InClusterConf should be used.

3.4.2 yamlProcessor

The yamlProcessor module consists of several methods that can manipulate yaml configuration files. These methods build a complete yaml read and edit system. This module is implemented in yaml.go. Each method is described as follows:

Table 1 Methods in yamlProcessor module

Method	Description
ChangeYaml	Modify a yaml file based on input parameters.
GenerateModelYaml	Generate a yaml file based on a device model object.
GenerateDeviceYaml	Generate a yaml file based on properties of a device instance.
GetModelProperties	Get a model property list from a yaml file
GetModelName	Get a model's name from a yaml file
mapChange	Internal function that automatically matches and modifies a Map object

Based on the yaml file reading and modification, the system can easily obtain or modify the system state. In addition, the module contains a structure “Property” that records the properties of the model.

3.4.3 K8sClient

This module contains a K8sClient class and several parsing functions. K8sclient obtain system information through the configProcessor module and access to KubeEdge to complete most operations supported by Kubeneters. The remaining parsing functions are utility functions that are used to obtain information that is difficult to get through the Client-Go native methods. They are shown in the following figure:

Table 2 Methods in K8sClient module

Method	Class	Description
NewK8sClient	K8sClient	Constructor, which generates a new K8sClient

		object according to the configuration.
GetPodList	K8sClient	Get a pod list from cluster, based on the namespace.
GetPod	K8sClient	Get a pod information based on the pod name and namespace.
DeletePod	K8sClient	Delete a pod based on the pod name and namespace.
DeployApp	K8sClient	Deploy a k8s app based on the namespace and yaml file.
DeleteApp	K8sClient	Delete a k8s app based on the app name and namespace.
GetAppList	K8sClient	Get the app list based on the namespace.
GetNodeList	K8sClient	Get the node list based on the namespace.
GetTimeStampFromDevice	None	Parsing a reported timestamp from a device instance object.
GetNodeStatus	None	Parsing a reported state from a node object.
GetTwinFromDevice	None	Parsing a device twin value from a device instance object.

3.4.4 dynamicClient

The DynamicClient module is implemented in the dClient.go file and currently has only one function applyYaml. This function generates a dynamic K8S client. From this client, the function can apply any yaml configuration file.

3.4.5 crdClient

The crdClient module is mainly responsible for the processing and control of crd resources. Specifically, several methods of K8sClient are implemented in the crdClient module, all of which are related to KubeEdge's device resources.

Table 3 Methods in crdClients module

Method	Class	Description
GetDeviceList	K8sClient	Get a device instance list based on namespace.
GetDevice	K8sClient	Get a device instance object based on name and namespace.
AddDeviceFromYaml	K8sClient	Add a device instance from a yaml file
DeleteDevice	K8sClient	Delete a device based on name and namespace
GetDeviceModelList	K8sClient	Get device model list based on namespace
GetDeviceModel	K8sClient	Get device model information based on model name
DeleteDeviceModel	K8sClient	Delete a device model based on name and namespace
AddDeviceModelFromYaml	K8sClient	Add a device model from yaml file
SetDeviceTwin	K8sClient	Set device twin for a device instance

3.4.6 Controller

To achieve more complex functionality, Controller further encapsulates the underlying interface and defines the data structure of multiple entities in the system.

In order to handle devices and models, control nodes, and add and remove services, we need to model these entities first. When an entity is processed, it is actually the modification of the metadata. The ResMeta class is first defined as the base class for all resource entities' metadata, and then multiple specific subclasses are derived from this class. The specific UML diagram is shown below.

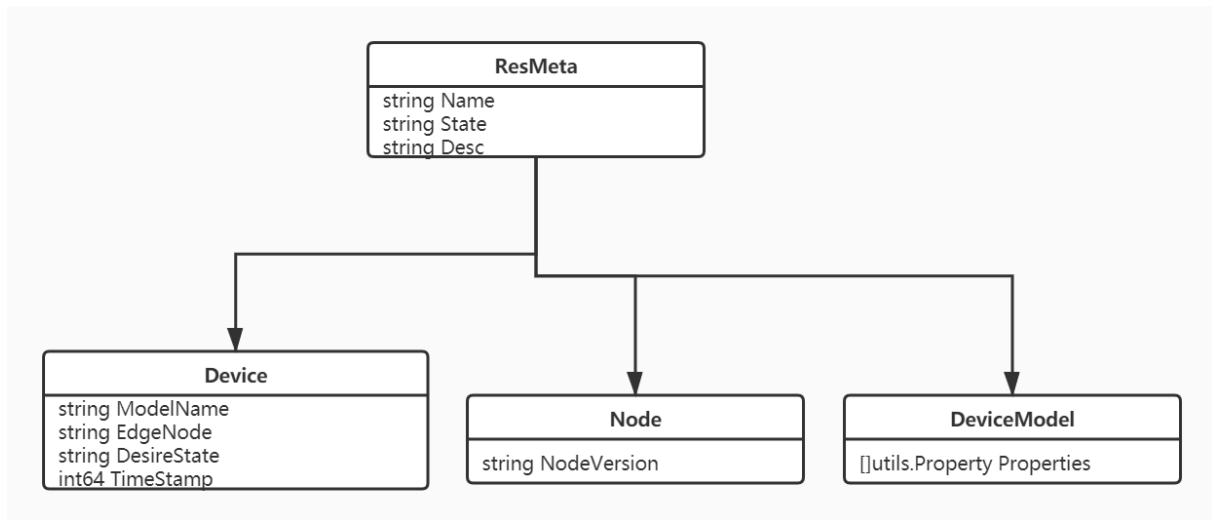


Figure 12 UML diagrams of device models, device instances, and nodes

The Controller class contains a K8sClient object. When you create a Controller, you need to specify a namespace, which means that a Controller can only serve one namespace. The implementation of this class is as follows:

Table 4 Methods in controller module

Method	Description
NewController	Create a new controller object
GetDeviceList	Get device list
GetDevice	Get a device info based on name
GetNodeList	Get node list
GetDeviceModelList	Get device model list
AddDeviceModel	Add a device model based on a model object
AddDevice	Add a device based on a device object
ChangeDeviceTwins	Change a device twin based on key-value pair

It looks like the methods in Controller are the same as K8sClient, but the input and output data are both higher level encapsulation. This encapsulation and aggregation hides a great deal of detail from the upper-level approach, allowing developers to develop controller-based services without knowing the KubeEdge principle.

3.4.7 Template

In some services, resources can be divided into different categories. Clients, for example, can be divided into mobile, laptop, or high-performance servers. At this point, if we want to handle a particular device, we can use templates to do so. Developers can define methods in the Template module that return different resource templates for Controller processing. In this project, `NewLearnerDeviceModel` method will return a `DeviceModel` object, which contains the information and initial value of all device twin needed by Learner.

3.4.8 Services

Applications built on controllers are called services. These services can run in the cluster or separately outside the cluster. In this project, the core service is called Flake, which is Federated Learning on Kube Edge. This service includes demos, system state retrieval and modification, device connection and federated learning under several different models, etc. Specifically, the Flake service can receive keyboard input, and the user can use commands to get information about the status of the current system, such as the time of the device's access, its current status, and so on. Flake can also use `DevicetTwin` to exchange data with devices for federated learning.

Like this service, other developers can make many different services that don't even use federated learning. This gives the system excellent scalability.

3.5 Implementation of Federated Learning

After completing the cloud control system, we need to build a specific service for federated learning on top of it. In order to achieve the flexibility of the system, most programs run in the KubeEdge cluster and are scheduled by the cloud system. However, this also increases the complexity of the system. Scheduling, storage location, and cross-platform are all need to be considered. To make development easier, instead of designing a federated learning framework ourselves, we implemented most of the functionality with the Flower framework, which was heavily customized and modified. The system structure diagram for federated learning is shown below:

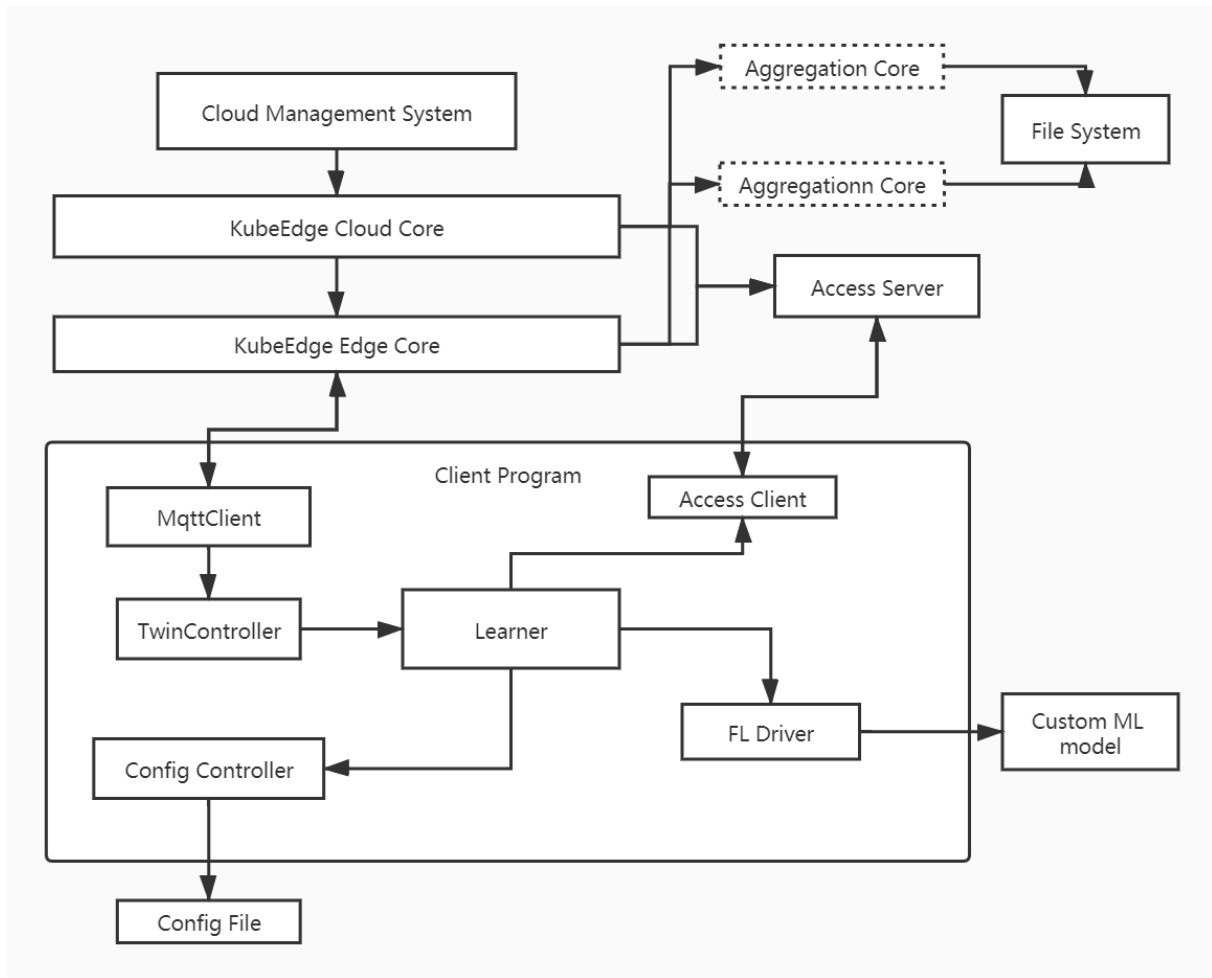


Figure 13 The structure of edge and client

The federated learning system can be divided into two major parts: Client and Aggregation Core. Client is a system that runs on the user's local device, maintains its own state, implements automatic connection edges, exception exit recovery, and accepts instructions for federated learning. Aggregation Core is actually the server side of the Flower framework for aggregation. It is packaged as a Docker image for system scheduling. In addition, there are auxiliary systems, such as Access Server, that authenticate and access the device.

3.5.1 Client

The Device class is a generic base class that handles connections, instruction uploads, issues, and persistence of the current device state. When Client is running, it will start an event loop. When an exception occurs or an instruction is received, the event loop pauses and the Client executes a specific action. The actions of Device can be divided into initial connection, state restoration, active state change and passive state change.

Initial Connection

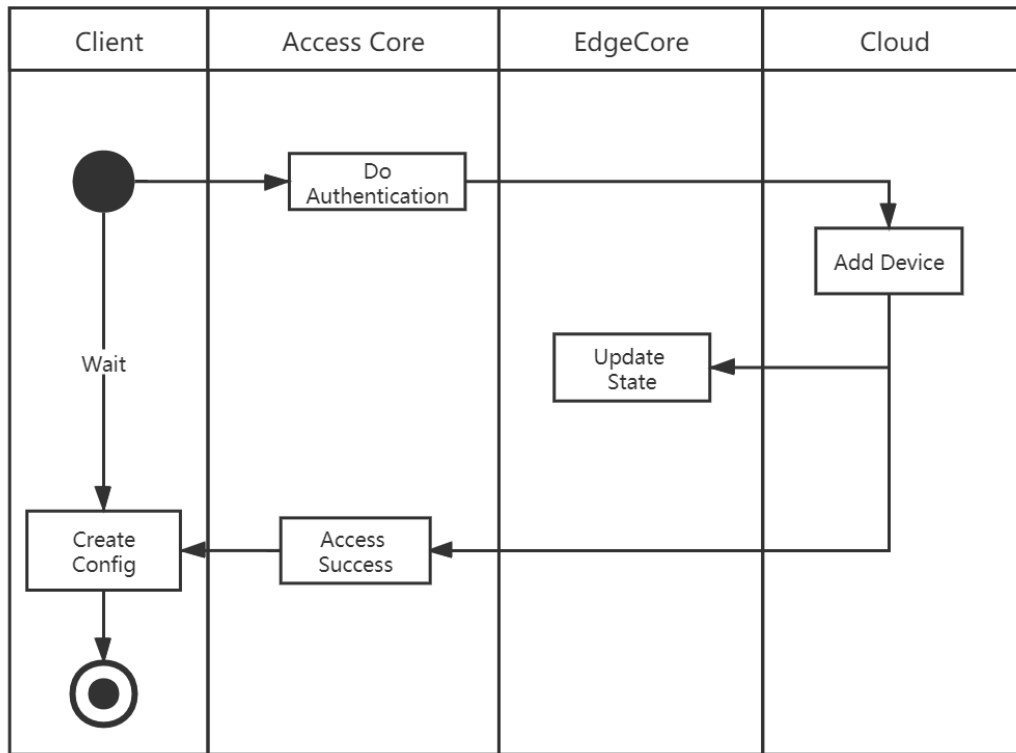


Figure 14 Flowchart of the first connection

On the first connection, the Client proactively looks for the Access Core and sends an HTTPS request with a key-value pair to verify its identity. Upon successful authentication, Access Core sends instructions to the cloud server to add devices. KubeEdge then tries to synchronize the system state to each component. When the Client receives a reply that the connection was successful, it adds the configuration file and proceeds with the rest of the work.

State Restoration

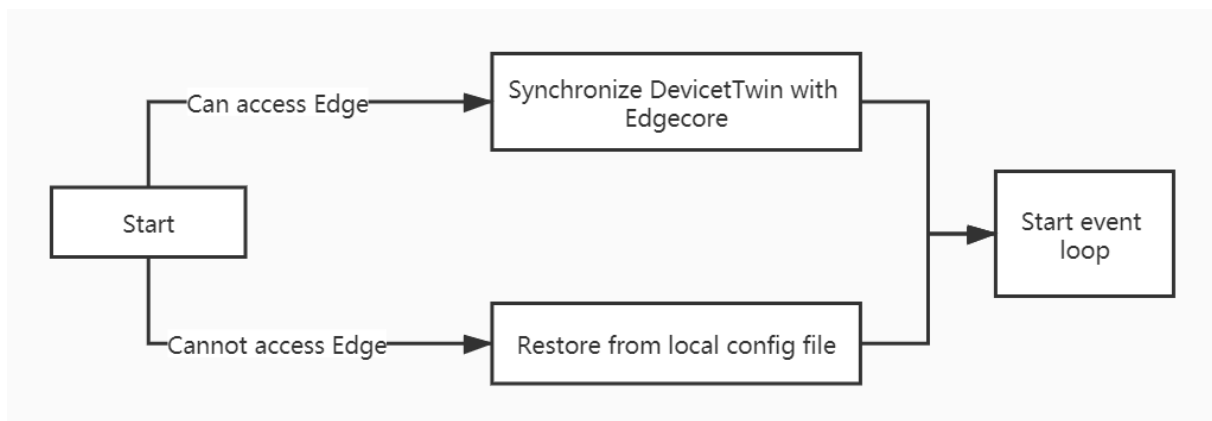


Figure 15 Flowchart of state restoration

When the Client starts, it first tries to send a request to the corresponding MQTT broker to see if EdgeCore is online. If EdgeCore is accessible, the Client synchronizes with DeviceTwin to restore its state. Otherwise, The Client recovers the state from the local config file.

State Change

The state changes of a Client can be classified as active or passive. To implement the state change, the Client has a full-duplex MQTT Client that communicates with EdgeCore.

Before discussing state changes, first discuss the priorities of the different components. The higher the priority of the component, the freer it is to modify its state. In this system, the highest priority is the cloud management system, which can modify any state at will, and other components need to synchronize with it. When the cloud management system is offline, the edge has some autonomy. At this time, the Client has the highest priority, and the edge node needs to synchronize data with the Client until the cloud is online again.

The active state change is easy to understand, where a Client sends an MQTT request to EdgeCore to modify DeviceTwin and synchronize it to the cloud. When an MQTT request is properly responded to, the Client persists the state ahead of time. If an MQTT request fails, the Client does not modify the local state, but rather waits for EdgeCore to issue a directive to make a passive state change.

Passive state synchronization is actually a form of command issuing from the cloud. When a directive needs to be changed in the cloud, DeviceTwin will be changed directly, and the change will be synchronized to the edge node and sent to the Client via MQTT request. The Client accepts this state change unconditionally and responds to it.

3.5.2 Learner

Learner is a derivative of Client that is responsible for Federated Learning process control and state transition. In order to achieve Learner, the following state should be defined first:

Table 5 The meaning of learner's states

State	Description
NotReady	Without the first connection, the system is in the initialized state
Online	Already connected to KubeEdge
Offline	Unable to connect to EdgeCore

ReadyForLearning	Ready to start Federated Learning
Running	In the Federated Learning process

Based on the state machine of Client class, Learner can easily maintain these states. When persisting the state, Learner uses the form of file, by creating special files of different filenames, to determine the current state, so that the state can be restored to normal under abnormal circumstances. In addition, Learner will control FL_Driver to achieve data reading and federated learning start.

3.5.3 FL_Driver

This component that drives federated learning is implemented in the FL_Driver class. The implementation of FL_Driver is very simple, and at its core is a function that starts a user-defined machine learning model in the form of a pipe. After federated learning is started, the driver will always query the running status of machine learning in a polling way and report to Learner in real time. In addition, the component will also check the file under the directory, and through the file to communicate with Learner, so Learner can asynchronously stop or start Federated Learning.

3.5.4 Custom ML Model

With the flexibility of the Flower framework, users can create their own machine learning models or change the centralized machine learning model to federated learning. Users only need to implement the training and verification process of machine learning, then they can easily access the platform. The platform also supports various general deep learning frameworks, such as TensorFlow and PyTorch.

3.5.5 MQTT Client

This module is responsible for communicating with KubeEdge via the MQTT protocol. The program is based on the PAHO.MQTT framework and overloads several important methods, such as on_message and message_control, to achieve full duplex MQTT communication.

3.6 Implementations of Different Models

All the three models mentioned above can be implemented in the system through the cooperation between the control of the cloud management system and the client. Specifically, the scheduling strategies and implementations of aggregation cores of the three models are

different, and the cloud-edge-client behaviours are also different.

3.6.1 Model-1 and Model-2

Model-1 is the simplest model, so its implementation is not complicated. Specifically, when training is needed, the cloud management system will first deploy Aggregation Core to the edge server. The client's deep learning driver then starts the machine learning client, attempting to connect to Aggregation Core to complete the machine learning. When the machine learning is completed, the data will be saved at the edge, and the cloud server can get the data through the interface provided by KubeEdge.

Model-2 is almost identical to Model-1, except that Aggregation Core will be deployed on a cloud server. When training is needed, KubeEdge dispatches an image containing only the deep learning driver to the edge server, which then collects the user data and trains it.

3.6.2 Model-3

Because Model-3 contains hierarchies, it has two Aggregation Cores. Specifically, when federated learning is required, the system first deploys an aggregation core in the cloud. When the cloud Aggregation Core is started, the system will schedule a special edge service on each edge node in turn, which includes an Aggregation Core and a lightweight client. After every k_2 aggregation, the edge service will output the data to the lightweight client by file system, which will upload it to the cloud server's Aggregation Core for processing. The client of this model is exactly the same as the client of Model-1, and is controlled by the cloud management system. After completing the training, the data is stored directly in the cloud.

The model implementation is shown in the figure below:

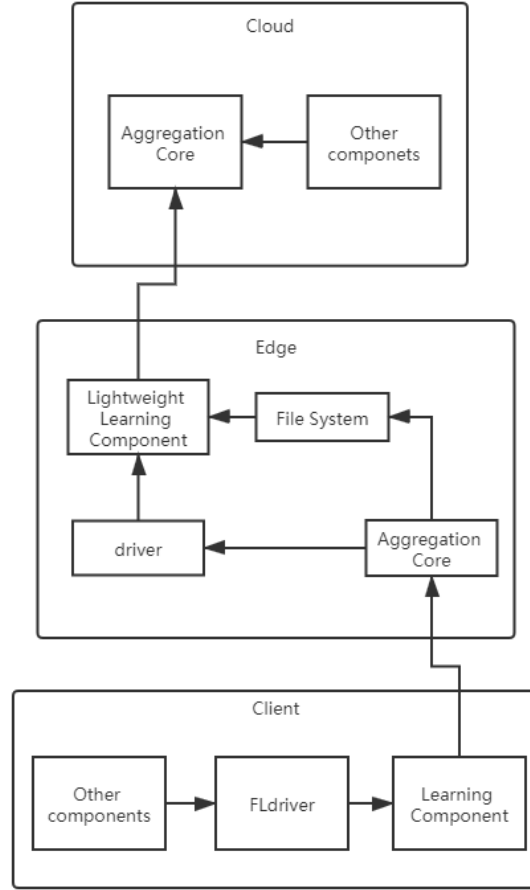


Figure 16 Implementation of Model-3

3.7 Security and Certification

The original intention of federated learning is to conduct machine learning without data exchange in order to fully protect the privacy of users. Therefore, in this project, in addition to relying on the security module of KubeEdge itself, we also added some additional ways to protect the system security.

3.7.1 SSL/TLS

According to the paper by Google, user data can be reverse-derived from the user gradient of Federated Learning using adversarial generation networks. Therefore, when federated learning is going, the gradient uploaded by the user also needs to be encrypted. We used secure and widely used SSL/TLS encryption.

Specifically, the Flower framework uses the HTTP protocol for communication between the server and the client. In this project, HTTP is replaced with HTTPS by overloading and non-intrusive modification. The client has the server address and certificate, and the encrypted transmission of data can be carried out after the implementation of SSL handshake.

3.7.2 Device authentication

In order to improve the security of the system, we need to identify and verify the equipment. Only authenticated devices can access the system for federated learning, which prevents hackers from attacking the model. Specifically, we design a device verification method based on key-value pair. Each device has a unique ID (key) and a unique password value, both of which are determined before the application is installed, so they can be stored on a cloud server or edge server ahead of time. When a device is connected to the system, it first sends an HTTPS request to the access module to try to verify its own key pair. The access module will query the local file and allow access to KubeEdge if the key content matches, otherwise it will deny the connection.

Chapter 4: Result and Discussion

After completing the design and implementation of the system, we used Raspberry Pi and several laptop computers to carry out practical tests. For three different federated learning models, the system shows good adaptability and performs various functions normally. In addition, we have also done some exception handling tests, and found that the system can handle some exception cases independently.

4.1.1 Devices Information

This is shown in the following table:

Table 6 Device Information

Device	OS	IP	Description
Lenovo Laptop	Ubuntu 16.04	192.168.31.172	20GB of RAM. For emulating multiple clients
Asus Laptop	Ubuntu 20.04 (VirtualBox)	192.168.31.227	12GB of RAM. Cloud Master
RaspberryPi 3B+ (A)	Debian 8	192.168.31.10	Edge Node
RaspberryPi 3B+ (B)	Debian 7	192.168.31.96	Edge Node
Mi Router	MiWifi OS	192.168.31.1	Router

4.1.2 Result of Model-1 and Model2

To verify the performance of Model-1 and Model-2, we ran tests on the PyTorch using the CIFAR-10 dataset (CIFAR-10 and CIFAR-100 datasets, 2021). The neural network structure is shown in the figure below:

```
def __init__(self) -> None:
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 6, 5)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 16, 5)
    self.fc1 = nn.Linear(16 * 5 * 5, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-1, 16 * 5 * 5)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Figure 17 Neural network structure

The specific test results are shown in the figure below:

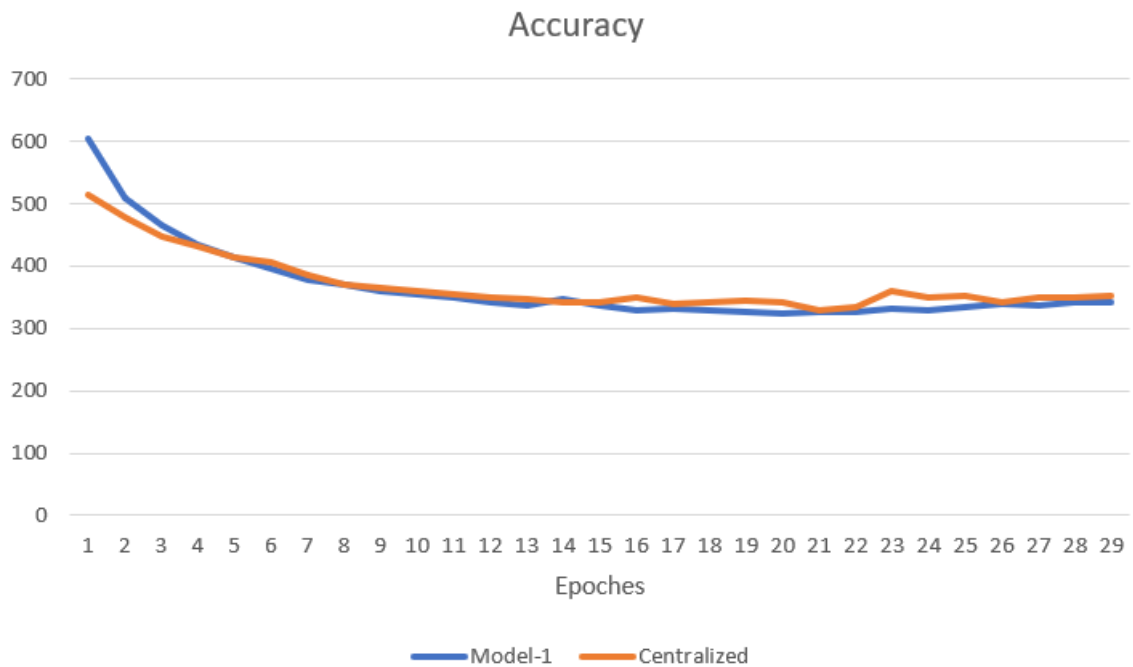


Figure 18 Accuracy of Model-1 and centralized deep learning

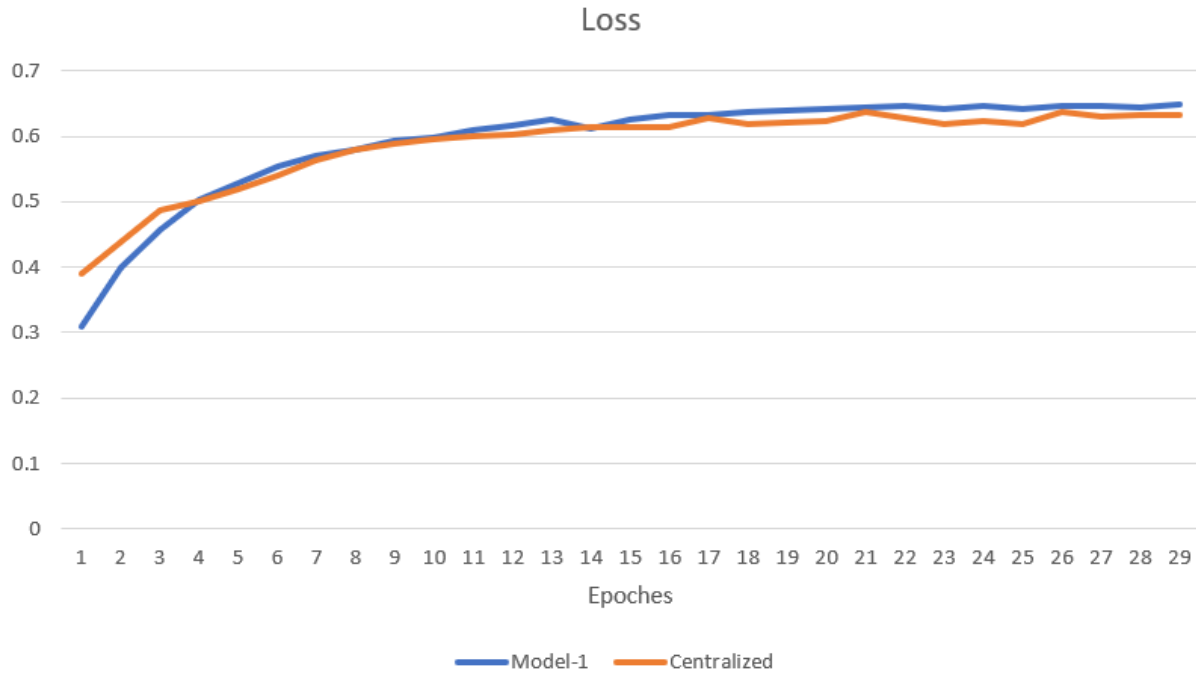


Figure 19 Loss of Model-1 and centralized deep learning

It can be seen that when using the same data set, the Model accuracy and loss of Model-1 are similar to that of general centralized deep learning. In the whole test process, the system shows a good stability. The test progress has been recorded and uploaded to a video website: <https://www.bilibili.com/video/BV1K54y1Y7JT>

The testing process of Model-2 is similar to that of Model-3, except that the training part of the machine learning is carried out on the edge server. Therefore, there is no need to repeat it here.

4.1.3 Result of Model-3

In the test of Model-3, due to the limitation of the hierarchy, I needed to simulate at least four devices to act as clients. As a result, my computational resources are insufficient and I can no longer use CIFAR-10 and the neural network as a test set. Finally, I used MNIST data set and a simple three-layer fully connected network to complete the test of Model-3 in this project.

The results of the Model 3 have been surprising. With the same data set, the accuracy and loss of Model-3 are similar to that of the very simple Model-1. Thus, this suggests that the performance of Model-3 is very close to that of centralized machine learning. In the process of testing, the system did not crash, delay and other phenomena, the service quality is very high. The following two figures show the results:

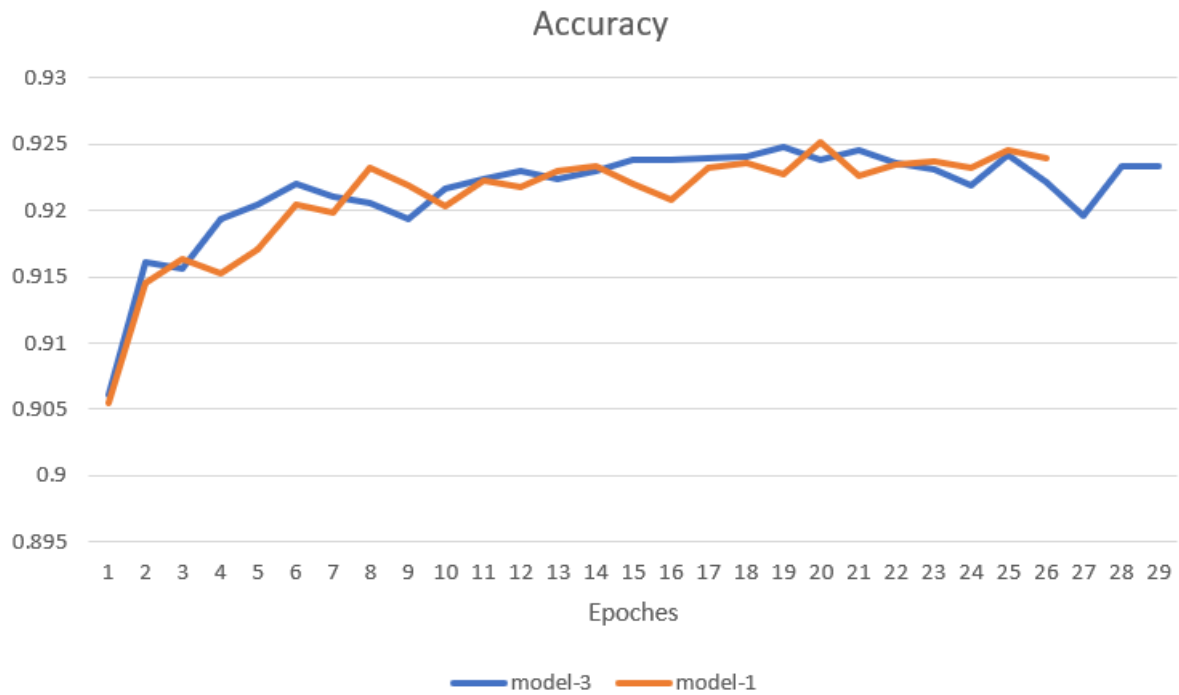


Figure 20 Accuracy of Model-3 and Model-1

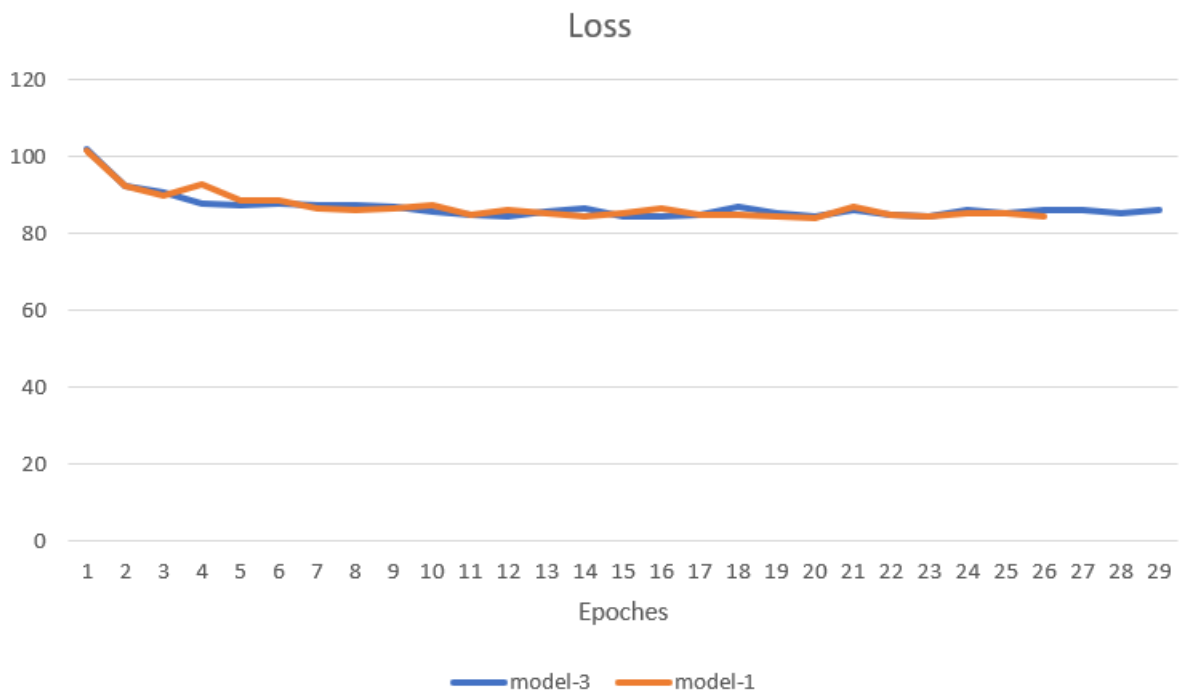


Figure 21 Loss of Model-3 and Model-1

4.1.4 Reliability Test Result

In order to test the reliability of the system, we designed a series of extreme cases to try to make the system recover from the error, and got good results. As shown in the table below:

Table 7 Reliability test result

Case	Result
During Model-1 training, shut down the cloud server	The training proceeded normally until completion
During Model-1 training, shut down one client	Other clients continue to train. Client that have been shut down and restarted will automatically reparticipate in training.
During Model-1 training, shut down the edge server	The devices immediately stop training and reset status. After the edge node is restarted, it can still obtain the data that has been trained before.
Close the edge node immediately after the command is issued by the cloud management system	Devices maintain original state. After restarting the edge node, the device receives the command
Set the wrong deep learning framework configuration for the device	The device cannot be trained, but it remains in its original state
Increase the network packet loss rate to more than 50%	The command was successfully sent from the cloud after several retries. The training speed of federation learning became very slow.

Chapter 5: Conclusion and Further Work

As a novel distributed machine learning framework, federated learning can play an important role in privacy protection, breaking data silos and other important issues. Edge computing is a computing mode that can give full play to the advantages of distributed computing. By combining federated learning with edge computing, the advantages of federated learning can be brought into play to some extent, and the cost can be reduced and the consumption of resources can be saved.

This project uses KubeEdge edge computing platform, combined with Flower framework, and successfully implements a flexible, extensible and robust Federated Learning platform. The platform can be divided into three parts: cloud, edge and local client. The cloud is implemented by Client-Go and is responsible for the management of the whole system. Edge is a very lightweight system that is cloud-based. The client is implemented in Python, and the machine learning algorithm can be customized by the user.

In terms of system reliability, the edge part of the system has a certain autonomous ability. When the cloud goes offline, the edge tries to maintain its current state. When both the cloud and the edge have problems, the local system can still boot up for data collection and other operations.

As for the performance of the system, the results of previous tests have been given. The result of federated learning is similar to the result of centralized operation, which can reflect the excellent performance of the system.

In terms of security, besides the built-in security authentication function of KubeEdge and Kubeneters, the system also adds many other security schemes by itself. For example, during federated learning, the system encrypts all the transmitted data through TLS/SSL. When the device is connected for the first time, the system also adopts TLS for encryption, and uses key-value pair for authentication.

In the aspect of system scalability, the system adopts the modular and extensible design from the beginning of the design, and sets aside the interface for the subsequent iteration and upgrade. For example, in the cloud control application, developers can customize their own services and explore more possibilities.

In addition, the system also has a lot of room for improvement. For example, more forms of secure encryption, such as hardware-specific TrustZone and SGX technologies, can be used for

trusted edge computing. In order to improve the ease of use, a graphical operation interface can be deployed in the cloud to help users use the system more conveniently.

References

Cs.toronto.edu. 2021. CIFAR-10 and CIFAR-100 datasets. [online] Available at: <http://www.cs.toronto.edu/~kriz/cifar.html> [Accessed 30 April 2021].

Kairouz, E. and McMahan, H., 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning*, 14(1).

Liu, L., Zhang, J., Song, S. and Letaief, K., 2020. Client-Edge-Cloud Hierarchical Federated Learning. *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*,.

Liu, T., Luo, R., Xu, F., Fan, C. and Zhao, C., 2020. Distributed Learning Based Joint Communication and Computation Strategy of IoT Devices in Smart Cities. *Sensors*, 20(4), p.973.

LUO, J., WU, W. and YANG, M., 2011. Mobile Internet: Terminal Devices, Networks and Services. *Chinese Journal of Computers*, 34(11), pp.2029-2051.

McMahan, B., Moore, E., Ramage, D., Hampson, S. and Arcas, B., 2021. Communication-Efficient Learning of Deep Networks from Decentralized Data. [online] PMLR. Available at: <http://proceedings.mlr.press/v54/mcmahan17a.html> [Accessed 30 April 2021].

Naimi, A. and Westreich, D., 2014. Big Data: A Revolution That Will Transform How We Live, Work, and Think. *American Journal of Epidemiology*, 179(9), pp.1143-1144.

Shi, W. and Dustdar, S., 2016. The Promise of Edge Computing. *Computer*, 49(5), pp.78-81.

Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L., 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5), pp.637-646.

Xiong, Y., Sun, Y., Xing, L. and Huang, Y., 2018. Extend Cloud to Edge with KubeEdge. 2018 IEEE/ACM Symposium on Edge Computing (SEC),.

Yang, Q., Liu, Y., Chen, T. and Tong, Y., 2019. Federated Machine Learning. ACM Transactions on Intelligent Systems and Technology, 10(2), pp.1-19.

Yann.lecun.com. 2021. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Available at: <<http://yann.lecun.com/exdb/mnist/>> [Accessed 30 April 2021].

Acknowledgement

In fact, this project is quite challenging for me. At the beginning, I didn't know how to start. I was very confused. I am very grateful to my tutor, who gave me directions for the implementation of the project and provided me with a lot of materials. I also want to thank the developers in the KubeEdge community for their help.

Risk and environmental impact assessment

In this project, due to the large amount of hardware and software coding involved, there are certain risks. In this section, we will analyse the potential risks of various aspects of the project and present their impact.

Specifically, problems can be divided into software problems and hardware problems. The possibilities and effects are shown in the following table

Table 8 Risk analysis

Risk	Likelihood Level	Consequence Level	L*C	Rating	Current Action
KubeEdge collapse	1	2	2	Low Risk	The client side includes a fault trap mechanism that can handle this situation
Network connection error	3	1	3	Low Risk	The client side includes a fault trap mechanism that can handle this situation
Client file system error	1	1	1	Low Risk	We can restart the client and try to resolve the error
Cloud server or edge module crashes	2	2	4	Moderate Risk	Kubeedge will try to fix the problem
AI training module error	3	0	0	No Risk	Just retrain
Docker collapse	4	1	4	Moderate Risk	KubeEdge can redeploy the images to each edge node
Edge node hardware crash	1	3	3	Low Risk	Technicians may be required to actually fix

Federated Learning Platform Design for Edge Computing

					hardware problems
Cloud server hardware crash	1	3	3	Low Risk	Technicians may be required to actually fix hardware problems