electronics

MDPI

Article

LACE: Low-Cost Access Control Based on Edge Computing for Smart Buildings

Haifeng Huang †, Hongmin Tan †, Xianyang Xu *, Jianfei Zhang and Zhiwei Zhao

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

* Correspondence: 202112081336@std.uestc.edu.cn

† These authors contributed equally to this work.

Abstract: With the explosive growth in personal mobile devices, offloading computation through nearby mobile user devices as opportunistic edge servers to support complex applications with limited computation resources is receiving increasing attention. In this paper, we first establish the optimal opportunistic offloading problem using the statistical properties of user movement speed and CPU load of mobile edge servers. We then determine the amount of computation to be offloaded to individual mobile edge servers. Moreover, we design an adaptive mechanism based on PID to realize the function of continuing large data packets from breakpoints, mainly used to adjust the size of data packets automatically. It efficiently avoids data loss and reduces the cost of resources through the latency deviation as the variable of the gain function to estimate the packet size. Finally, an access control system based on edge computing is designed and developed to make full use of the mobile phones of nearby users. It can address the shortcomings of traditional schemes with high latency to some extent, and it makes latency lower and data reliability higher.

Keywords: low cost; edge computing; smart city; system optimization

Citation: Huang, H.; Tan, H.; Xu, X.; Zhang, J.; Zhao, Z. LACE: Low-Cost Access Control Based on Edge Computing for Smart Buildings. *Electronics* **2023**, *12*, 412. https://doi.org/10.3390/electronics12020412

Academic Editors: Antonio Brogi and Juan-Carlos Cano

Received: 21 November 2022
Revised: 2 January 2023
Accepted: 9 January 2023
Published: 13 January 2023

1. Introduction

Intelligent buildings are based on the integration of modern advanced technology on the basis of traditional buildings, so that the system, structure, management, and services of buildings can be optimized and combined according to people's needs. They embody the humanization of architecture and create a comfortable, convenient, and efficient environment for people. In addition, with the world's energy crisis today, architects, engineers, and building managers must make buildings energy-efficient and intelligent in terms of function and use. Technologies applied to smart buildings will improve the built environment and the occupants' functionality, and the reduction of operating costs is imminent [1].

The access control system is an essential part of the intelligent building sector, namely the Access Control System, or ACS, for short. It usually refers to a control system that uses modern electronic and information technology to manage and restrict the entry and exit of people or things at the entrance and exit passages, including granting or denying access, recording, alarm, and so on. The current design of access control in intelligent buildings is mainly about transmitting images to the cloud platform server, which processes the images and returns the result. The access control system primarily consists of a terminal access control node, a cloud platform server, and a remote control platform, which accesses the Internet. Thus the API of the cloud platform, through communication means such as WiFi, 4G, and 5G, sends the collected data to the server. The terminal access control node receives and responds to commands from the remote control platform [2,3].

The above solutions still have certain shortcomings [4]:

- There is the problem of insufficient local bandwidth and high latency caused by network congestion.

- There are specific privacy and security issues with the cloud platform solution, as the access control system may involve private premises data.

The European Telecommunications Association introduced the concept of mobile edge computing in 2014 to meet users' specific application and service needs in terms of real-time, agility, security, and privacy protection by enabling functions such as data caching and computing at the edge of mobile device networks. As edge computing nodes can be deployed at the edge of the network, they have the potential to overcome the barriers of traditional central clouds and the resource constraints of conventional mobile networks by being able to transmit data over the local network and, thus, not be affected by Internet congestion [5].

Due to all the above factors, we propose a novel architecture for low-cost access control for intelligent buildings. Compared with the traditional solutions, the proposed architectures allow ultra-low-cost front-end devices and pervasive edge computing based on smartphones. The low-cost camera can be placed anywhere to establish access control systems, eliminating massive deployment costs, for example, wiring, embedding, and other placement costs. Based on the architecture, we present the specific designs for front-end and edge-side devices. The front-end hardware cost is vastly reduced by reducing the path of data dissemination. Devices in the environment need more potential computing power; Ubiquitous and low-cost smartphones can be utilized as edge service providers. We implement LACE and evaluate its performance in terms of both identification accuracy and system latency. The experimental results show that LACE can achieve better results than the traditional solving methods.

2. Related Work

2.1. Traditional Intelligent Access Control System Solutions

To meet the requirements of security automation in intelligent buildings, intelligent access control is a vital link, but the current access control system generally uses digital IC cards, fingerprint recognition, password recognition, and other methods; There are many problems, such as low security, difficult maintenance, network bottleneck, and insufficient integration.

In 2014, Prof. Y Su led a team to develop a system to realize the authentication of the access control system and the mobile intelligent terminal through a pseudo-random sequence encryption algorithm, thus, allowing the control of the access control system by the mobile smart terminal [6]. However, due to the technical limitations of the Bluetooth connection itself, it is not easy to have a good solution for large-scale data transmission, especially in today's world of artificial intelligence, where data is massive and uninterruptible. This has led to this system being used in a relatively narrow range of trial scenarios.

2.2. Existing Edge-Based Solutions

With the rapid arrival of the Internet of Everything era and the popularity of wireless networks, various sensors and ubiquitous wireless networks provide a good foundation for constructing intelligent cities. Still, at the same time, it also brings a massive consumption of resources, the total amount of global data is greater than 40 zettabytes [7], and 45% of the data generated by the IoT (Internet of Things) will be processed at the edge of the network, the massive amount of data generated by smart cities will put enormous pressure on the centralized processing model of cloud computing and urban uplink bandwidth. Hence, the computing model for edge-oriented devices is also applied to smart cities. Then the concept of mobile edge computing is introduced [8].

Referring to the existing literature, we found similar research work, like a video surveillance system based on edge computing and permission blockchain proposed in the literature [9], which consists of three parts: a physical layer, a data service layer, and an application layer, enabling large-scale sensor information collection and data processing as well as storage. Our final implementation case is similar to what is mentioned in this

literature. Instead of using a Raspberry Pi as an arithmetic tool, we have used a more universal and convenient smartphone.

2.3. Main Contributions

Compared to all the previously mentioned methods, this system proposes a low-cost edge computing framework for ubiquitous environments. Most IoT devices serve human beings, and in modern society, smartphones often follow the operated object, so in this system, we choose to use smartphones as the core server of the edge service system and develop the APP on the smartphone. Our main contributions are summarized as follows:

- High compatibility, traditional intelligent home gateways or other devices are bound to their specific operating system. There are specific differences when different types of IoT devices exist in the environment, i.e., multiple other gateways are needed for service provision, but the current smartphone operating system is more uniform, with a significant market share occupied by Android, iOS, and Hongxing. Using a smartphone operating system as the underlying system development can significantly provide the compatibility of the service system.
- Low hardware device requirements, compared to other solutions. This system hardly needs to deploy additional computing devices and provides services entirely by utilizing existing idle computing resources.

3. Solution Design

Combining the characteristics of smart city construction and the particular advantages of edge computing, this paper proposes a low-cost intelligent building access control system based on edge computing, which can deploy the system in scenarios such as residential houses at a lower cost and achieve stable and reliable access control functions.

To realize security automation in intelligent buildings, the access control system needs to be able to meet the following requirements:

- The ability to capture and store biometric data of different users, such as faces and gestures.
- The ability to process, analyze, and differentiate data from different users so as to return different results for different users with different feature information, such as granting or denying access.
- The ability to transmit data securely and reliably between different terminals.
- The ability to perform self-processing and recovery from some task exceptions.

3.1. System Overview

To meet the above requirements, the access control system proposed in this paper is designed modularly, as shown in Figure 1. The whole system is divided into an IoT front-end device module, a task offloading node module, a mobile edge server, and a user interaction module. In addition, they need to be connected via the Internet for communication [10], as in Figure 2.

A complete access process can be expressed as shown in Figure 1. When a user approaches, the front-end device will collect and briefly store the user data, then communicate with the task offloading node and notify the task offloading node of the task type. At the same time, the task offloading node needs to search whether the user carries a mobile edge server and dynamically maintains a table of available mobile edge servers. Upon receiving a task request from the front-end device, it adds it to the corresponding task queue according to the task type. It selects the optimal set of mobile edge servers to assist in offloading and computation [11].

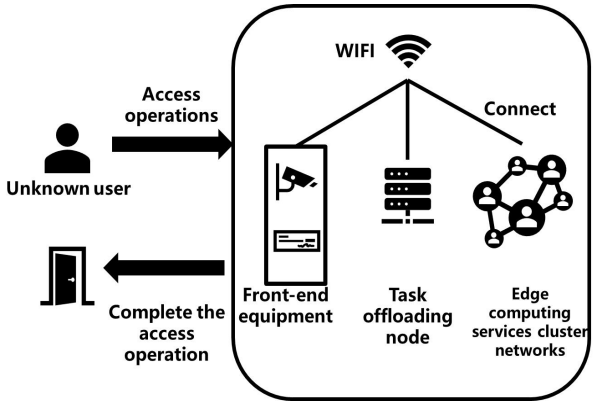

Figure 1. Design of access control system.

Figure 2. Diagram of IoT connection with smartphones.

3.2. Front-End IoT Device Design

The front-end equipment module of the intelligent building access control system proposed in this paper should perform the following functions.

- Capture and briefly save image data.
- Exchange data with task offloading nodes via WiFi, Bluetooth, or Zigbee technology.
- Execute the command issued by the mobile edge server after completing the calculation and perform the granting or denying access for the person requesting access.

The front-end device is designed as shown in Figure 3 and consists of an operation execution layer and an operation control layer. The lower layer is the operation control layer, including the task scheduler, communication module (WiFi module, Bluetooth module, Zigbee module or coexistence), I/O pins, and the underlying hardware. When multiple operations coexist, the task scheduler will queue up the upper-layer tasks according to the completion of tasks in the communication module, and then it will notify the upper layer through the I/O pins of the operation command to be executed (e.g., 1 for granting access, 0 for denying access). The upper layer is the operation execution layer, which includes the hardware necessary to operate, such as the gate switch, the display, the loudspeaker, the light, etc. A normal and complete workflow for a front-end device would be like this:

1 The infrared radar sensor detects an incoming object, and the camera acquires image data.
2 The task scheduler sends a task request to the task offloading node via the communication module.
3 The task scheduler acquires data from the sensor cluster via the communication module and submits the task data to the designated set of mobile edge servers based on the content returned by the task offloading node.
4 Based on the results returned by the set of mobile edge servers, an operation command is sent to the operation execution layer.
5 If the operation is successful, the task is completed; Otherwise, the same process is repeated.

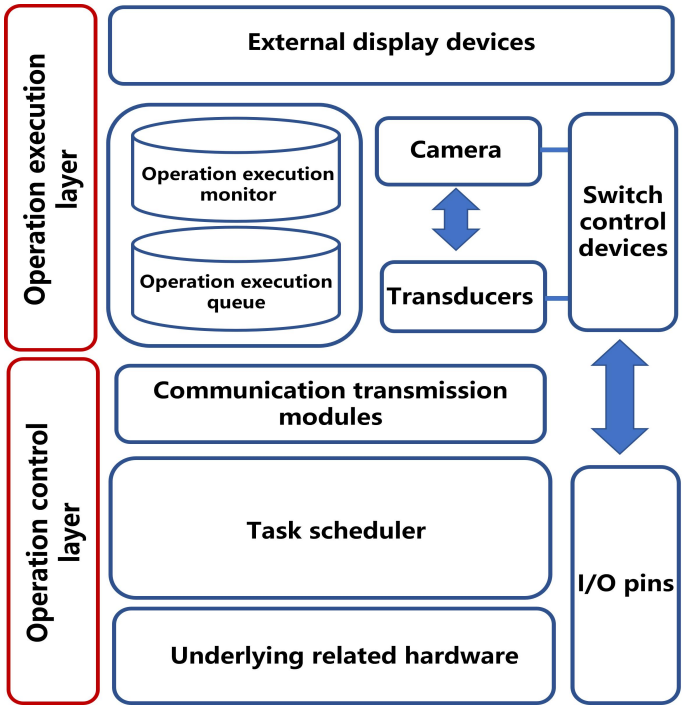

Figure 3. Front-end device architecture diagram.

3.3. Edge-Side Design

When edge computing was first proposed, it reflected a relative edge compared to cloud computing. Solutions to implement edge computing often use fixed edge gateways with specific computing resources, among which the better-known one is cloudlet [12], which can provide computationally intensive services for mobile devices, but compared to the traditional approach, this paper proposes a ubiquitous environment with a low-cost mobile edge computing server architecture. Most IoT devices serve humans, and in modern society, smartphones are often carried by humans, so in this system, we choose to use smartphones as the core servers of the edge service system. A dynamic service access mechanism can be realized by using smartphones in conjunction with task offloading nodes, and smartphones' computing power can be fully utilized under the task offloading nodes.

From the above literature and related studies, we have identified a design solution for the Edge-side. The mobile edge server side is still modular, dividing the app into communication and service provisioning modules. A clear UI module is also available.

3.3.1. Communication Module

The communication module has two modes: service access mode and service provision mode. The communication module needs to establish a communication connection with the task offloading node when the users bring their smartphone within the communication range of the task offloading node, which we do here via the service-type-aware DHCP protocol. Once the smartphone is successfully connected to the task offloading node, it needs to perform calculations on the offloading tasks and transmit information, including the current phone status and the built-in service model.

Design of communication protocol: As smartphones provide mobile computing resources in edge computing service networks, they need to be able to be accessed at any time. In some instances, errors such as hardware and processes can occur due to the sudden departure of users. Especially when the data size of the task is large, the result can not be returned due to such errors; some data have to be calculated repeatedly. We have chosen to develop a new communication protocol based on the DHCP network communication protocol. We call this PDHCP protocol. Using the IoT device as a server for task offloading,

critical computing tasks are partitioned into multiple pieces, and the data of each current piece are transmitted to the corresponding offloading nodes. Then the results of the calculations or command requests are returned step by step. A complete implementation of the algorithm is shown in Algorithm 1.

Algorithm 1: Data transfer algorithm

Input: Specify packet size M_{max}
Computing-Server side:
While (Calculation tasks not completed)
  Computing;
  Transmit data to IoT device.
END
IoT-Device side:
Transmit data to service nodes;
Receive the returned results from the service nodes.
if *timeout* **then**
  Change another service node;
  Transmit the last packet to the new service node.
Final

As shown in Figure 4, compared to the traditional DHCP protocol, PDHCP adds a segment of the data message to represent the marker number of that packet. Whenever the IoT device's calculation result packet is received out of time, a decision is made to reconnect the node based on that marker number and restart the calculation task at the breakpoint.

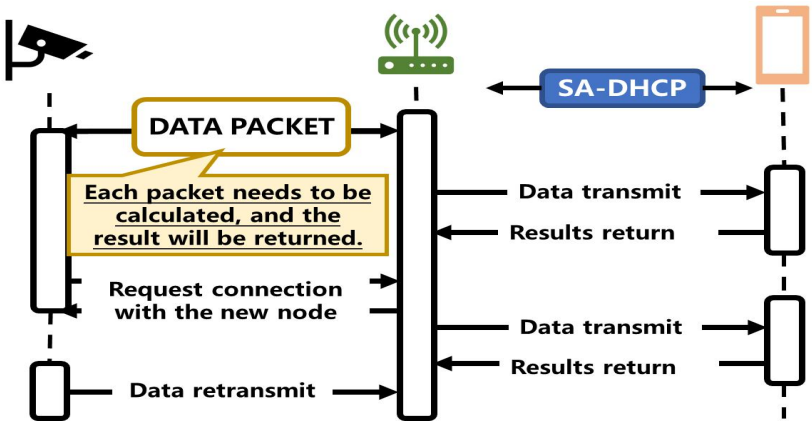

Figure 4. Schematic diagram of the PDHCP protocol.

Adaptive parameter adaptation mechanism: The protocol approach mentioned in the previous section requires a determination of the packet size. However, this also has specific problems: How to determine the size of the data sub-packet, if the packet is too small, repeated receiving and sending operations will cause greater cost consumption and do not meet our requirements to reduce energy losses; If the packet is too large and it will not effectively reduce the computational cost. Therefore, based on the mechanism of PID automatic control, we propose an algorithm that can readily adapt to the latency and data transmission requirements to calculate the appropriate packet size.

PID (Proportion Integration Differentiation) means proportional, integral, and differential control. The standard PID algorithm is generally used in high-precision scenarios requiring adaptive control, such as CNC machine tools. Still, because of its rapid regulation, there are ways to expand its application. According to the literature [13], they propose an algorithm for a fuzzy PID, a discrete-time version of the traditional PID controller that retains the same linear structure of the proportional, integral, and derivative parts with

constant coefficients but self-adjusting control gains. Or a scheme similar to the one mentioned in the literature [14]: A method that uses machine learning to generate controller tuning algorithms automatically. The process constitutes a decision tree that selects the rule that best improves the controller characteristics from a set of tuning rules. The decision tree is constructed using a set of trained example systems. The resulting tuning algorithm is evaluated using a large group of independently generated example systems. This also matches our experimental requirements, but finding a training set of potential strategies for each application scenario is not easy [15].

Here we choose a similar treatment, taking into account the requirements of practical application scenarios, and we end up with the latency deviation *err* as the variable of the gain function. The latency deviation *err* is calculated as shown below.

$$err(k) = O(differ(Actuallatency, Ideallatency)) \tag{1}$$

where the function O represents a method of converting latency to packet size, and its exact conversion is directly related to the actual network transmission speed. We will need to study and evaluate it more accurately in future work.

$$u(k) = Kp \cdot err(k) + Ki \cdot \sum_{i=0}^{k} err(k) + Kd \cdot (err(k) - err(k-1)) \tag{2}$$

where Kp, Ki, Kd are the three parameters for regulation and need to be initialized to determine.

It should be noted here that the values of these parameters, in general, will not affect the convergence of the gain. Still, a better selection of parameters will allow the function to complete its conditioning and convergence work more quickly. We believe that at this stage, the best values for them should be selected by some machine learning approach, which will be reflected in subsequent research work. In the literature [16], we can see how the PID parameters are tuned using BP networks, which is an excellent insight into our work. In the future, we may also use such an approach for optimizing and adjusting the parameters.

As shown in Figure 5, $u(k)$ represents the gain size of the kth transmitted packet and the size of the next packet is $P(k+1)$ can be expressed as

$$P(k+1) = P(k) + u(k) \tag{3}$$

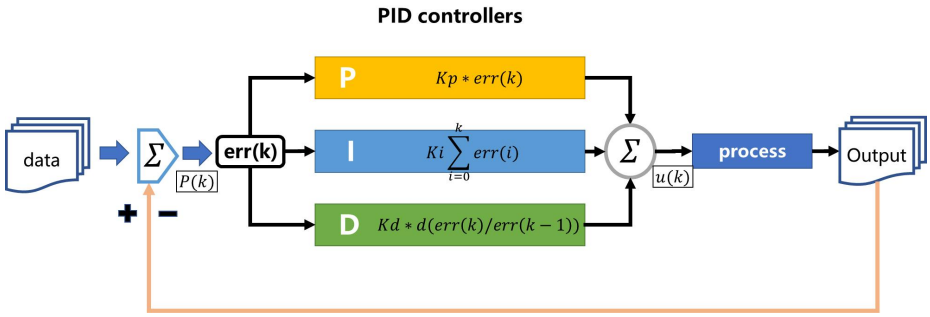

Figure 5. Diagram of the PID adaptive regulation mechanism.

In this process, as we do not need to granulate the packet size excessively fine, we need to fuzz the size of *err* and not change the packet size when its deviation size is in an acceptable range.

It is also important to note that this would only be used in cases where the scale of the computing task is large and a large amount of data transmission is required. The usual application scenarios include intelligent building access control systems with massive movements of people, and conducting research in this area could enhance the applicability of our proposed LACE.

3.3.2. Computing Service Module

According to the current mobile phone performance and user requirements, part of the deep learning model is pre-set in the app in advance. When a service is provided, the form of the result is determined by the service type. The results are transmitted to the front-end device through the communication module.

The communication module has two modes: service access mode and service provisioning mode. When users bring their smartphone within the communication range of a task offloading node, the communication module needs to establish a communication connection with the task offloading node, which we have done here by defining the PDHCP protocol. Similar to the service-type-aware DHCP protocol, once the smartphone has successfully connected to the task offloading node, it needs to be able to perform calculations on the offloaded tasks and needs to transmit information, including the current phone status, the built-in service modes, etc. [17].

3.4. Task Offloading Module Design

Using the Raspberry Pi 4B as a task offloading node, this system designs an efficient task offloading node module by explicitly considering the mobility, and heterogeneous resources (CPU, etc.) on the mobile edge server, which offloads the task to the most appropriate set of mobile edge servers, thus, minimizing the expected execution time. Usually, due to the need to perform complex tasks such as object detection on video captured by HD cameras in many scenarios where one edge computing node has limited computational power or is impacted by mobility, it may not be sufficient to complete the task efficiently. We consider chunking and offloading a video frame to a different server for parallel computation [18]. However, the task offloading problem becomes more challenging in a multi-mobile edge server scenario because of the variation in user mobility and mobile edge server resources, where each task offloading node can offload different amounts of task data to other mobile edge servers, and the computing resources of mobile edge servers is usually limited. To overcome the above challenges, we propose a system with mobile servers to determine to which servers each task of the front-end device is offloaded for more significant performance gain.

The main design features of this module are: (1) This module considers the impact of the user's movement speed change on the task calculation; (2) At the same time, the impact of the CPU load of the mobile edge server carried by the user on the task calculation is considered. A reliable computation and communication model is established based on the above features [19,20]. The execution time of task k of the front-end device at mobile edge server j can be calculated as

$$t_{(comp,k,j)} = \frac{\gamma_{k,j} s_k z_k c_j}{\Phi_j} \tag{4}$$

where $\gamma_{k,j}$ is the proportion of data size that task k of the front-end device allocated to mobile edge server j by task offloading node N, s_k is the size of the input data (in bits) of the task generated by front-end device k, z_k is the number of CPU cycles required to execute one bit of the task, c_j is the CPU load of mobile edge server j, and Φ_j represents the number of CPU cycles per second that mobile edge server j can execute.

The data transmission rate of the front-end device can be calculated as

$$r = W \log(1 + \theta) \tag{5}$$

where W is the bandwidth of the front-end device and θ is the SINR between the front-end device and the task offloading node. The data transmission rate between the task offloading node and the mobile edge server j can be calculated as

$$r_j = W_N \log(1 + \mu_j(c_j, v_j)) \tag{6}$$

where W_N is the bandwidth of the task offloading node, μ_j is the SINR between the task offloading node and mobile edge server j, and v_j is the movement speed of mobile edge server j. The offloading time of task k from the front-end device to the mobile edge server j can be calculated as

$$t_{off,k,j} = s_k / r_k + \sum_{j=1}^{J} \frac{\gamma_{k,j} s_k}{r_j} \tag{7}$$

The end-to-end latency between task k of the front-end device and mobile edge server j can be calculated as

$$t_{k,j} = t_{comp,k,j} + t_{off,k,j} \tag{8}$$

When a task is generated for the front-end device, the task offloading node selects the set of mobile edge servers whose latency is within the deadline to execute this task by evaluating the end-to-end latency between it and mobile edge server j.

4. Implementation

This section provides a detailed description of the specific implementation methods and processes of the service architecture section described earlier.

As shown in Figure 6, the image information is collected by the front-end device when the process starts. After getting the data to be processed, the front-end device requests the corresponding service through the task offloading node and waits for the result of the task offloading. After the server is allocated, the task type and data are transmitted to the given mobile edge server via an HTTP request. After the server successfully receives the task information and complete image information and passes the checksum, the data is sent to the calculation module for calculation. The calculation result is transmitted to the front-end device via HTTP response when the task calculation is completed, which completes the service process. If no mobile edge server is available, the task offloading node notifies the front-end device and waits for new available servers. In the event of data errors or loss during transmission, the mobile edge server communicates directly with the front-end device to complete the data transmission. After multiple failures, the process is restarted to allocate another server.

4.1. IoT Front-End Device Implementation

This module uses the ESP32-CAM, like in Figure 7, development board with integrated WiFi and Bluetooth modules from AI-Thinker as the primary development board for the front-end device, with an OV5640 auto-focus camera and an infrared sensor as auxiliary sensors for implementation. In addition, motors, LCD screens, and other devices are still required as hardware support for the operation execution layer.

This paper uses a third-party library, HttpClient [21], as the network framework for development, providing a RESTful API-like interface as a communication tool. When a task demand arises, first, the front-end device sends a POST request to the task offloading node with parameters such as service type, data size, data format, and other task information like the IP address or another kind of communication address of that mobile edge server. Then it waits for the task offloading node to return the offloading result.

The front-end device then establishes an HTTP connection with the mobile edge server to transmit the collected data. It maintains the connection waiting for the calculation result, and both parties shake hands to disconnect after receiving the result. After the front-end device is powered on and the camera is successfully initialized, it can work directly if connected to the task offloading node using a wired connection. Heartbeat packets must be sent to each other to keep them alive during the operation. In normal operation mode, each front-end device first applies a service provisioning request to the task offloading node and then requests service from the mobile edge server at a rate of 30 frames per second to complete the intelligent access control task after the mobile edge server is allocated and its IP address is obtained.

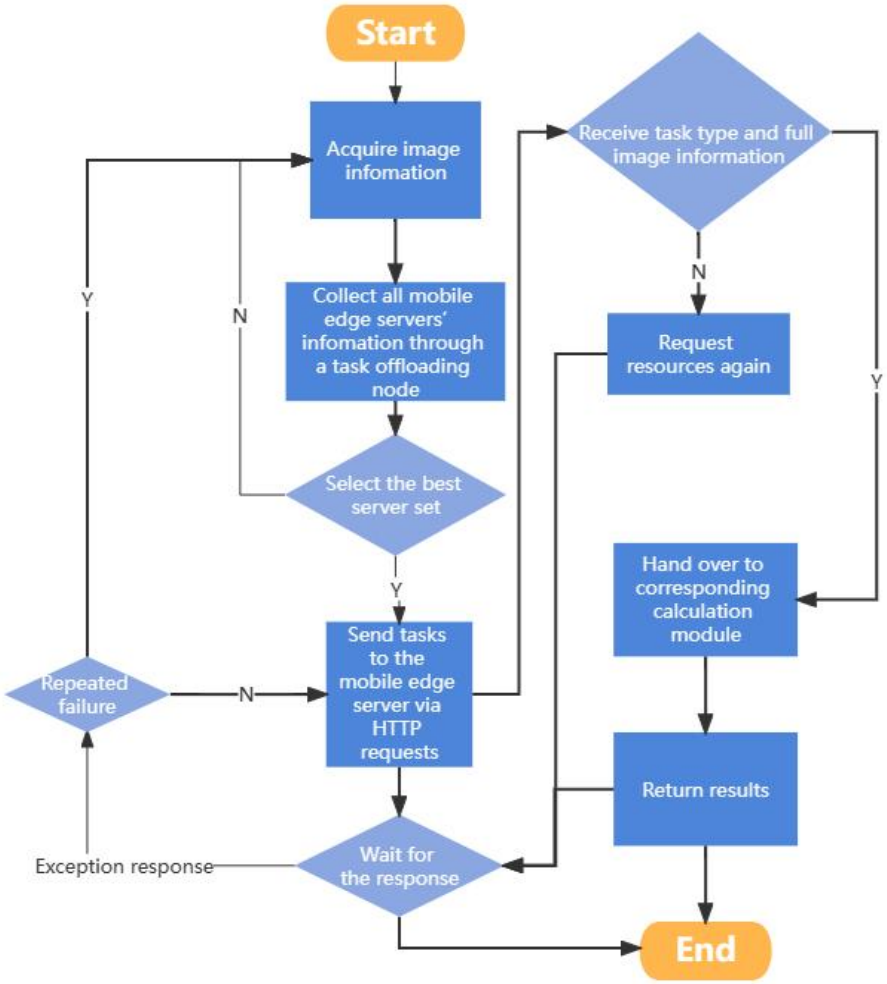

Figure 6. Mobile edge service provision process.

Figure 7. ESP32-CAM.

4.2. Communication Module

This module uses the OkHttp library [22,23], widely used by Android, as the implementation framework for the communication module. Its support for the HTTP/2 protocol

allows all requests connected to the same host address to share a socket, providing an excellent efficiency gain for small but frequent data transmissions from IoT devices. Secondly, the latency of requests can be reduced by the connection pool. A multi-thread model was used to implement the communication module during development. The task offloading node can offload multiple tasks to different mobile edge servers simultaneously, contributing to the system's overall efficiency. The edge servers used in this paper are Google Pixel 4 and Google Pixel 5 with Android 10 (API level 29) or higher. The mobile edge server will maintain a scanning frequency once every 500 milliseconds in the test environment. When the user carries the mobile edge server into the service request range of the task offloading node, the SSID corresponding to the task offloading node appears in the scanning result of the mobile edge server, and the service access mode will be performed automatically. The mobile edge server will include the device model, memory, load, remaining power, etc. The mobile edge server will register its basic information to the task offloading node and, at the same time, will maintain the connection to provide an interface for updating the device status. After completing the service access link, the communication module will enable multi-thread mode. There is always a thread listening for all source address packets under WiFi to accept tasks from the front-end device. When the data of the task is received, it will be handed over to the calculation module for processing according to the task type and corresponding requirements. At the same time, heartbeat packets are exchanged for connection maintenance, and the calculation results are returned via the original connection when the result callback function responds.

4.3. Calculation Module

Under the Android platform, this system uses the lightweight TensorFlow Lite framework for implementing AI algorithms. Its advantages are lightweight, low latency, and high portability, which works significantly under various models of smartphones with the Android operating system and facilitates subsequent porting to the IOS platform. We have implemented three deep-learning models built in.

4.3.1. Face Detection Module

Using FaceNet proposed by Google in CVPR2015 [24,25], the image is mapped to a 128-dimensional feature space by a deep convolutional network. In addition, L2 regularization is used to filter good features. It can perform face detection better while using a 1×1 convolutional form, which can reduce the number of parameters without decreasing the accuracy. By deploying this lightweight network to mobile devices via TensorFlow Lite, we can obtain a high confidence level with less computational power required and improve the model's accuracy while keeping the processing speed unchanged compared to traditional face detection.

4.3.2. Target Detection Module

Using the classical SSD-MobileNet lightweight target detection network, the SSD idea is used to generate prior frames, i.e., pre-selected target frames [26], while combining YOLO's regression idea and non-great suppression, and the depth-separable convolution is used to reduce the computational effort and perform computational slimming at the channel level. In the training process, positive and negative sample pairs are generated to improve the recognition of image background frames, which ultimately enables the whole network to achieve better face detection with low parameters. At the same time, the convolutional operation can reduce the computational effort of the model at the image channel level and achieve a good adaptation in low-power mode.

4.3.3. Gesture Recognition Module

The gesture recognition module is obtained using the MobileNet model pre-trained in the ImageNet dataset and trained in the publicly available gesture dataset using a migration learning approach. MobileNet was proposed by Google in 2017 [27]. MobileNet used depth-

by-depth convolution and point-by-point convolution compared to standard convolution, and the convolution kernel is split. This reduces the number of convolutional operations without changing the feature map channels. MobileNet's reduction in convolutional layers can significantly reduce the network's running time.

4.4. UI Module

As shown in Figure 8, the UI module connects the user to the mobile edge server so that the user clearly understands current device usage, combined with XML language development to build an interactive environment. The human–computer interaction concept of the UI is based on simplicity, providing a straightforward interface and visualization of the user's operations. In line with this concept, the operating logic of the UI is designed to be equally simple, and the app's main interface can be directly accessible by clicking on it. The operable part of the main interface consists of two buttons: Start, which controls whether the machine is used for the mobile edge server to provide services, and Show, which displays the last image captured by the front-end device and a description of the previously assigned computing task in the lower image window and text window, respectively.

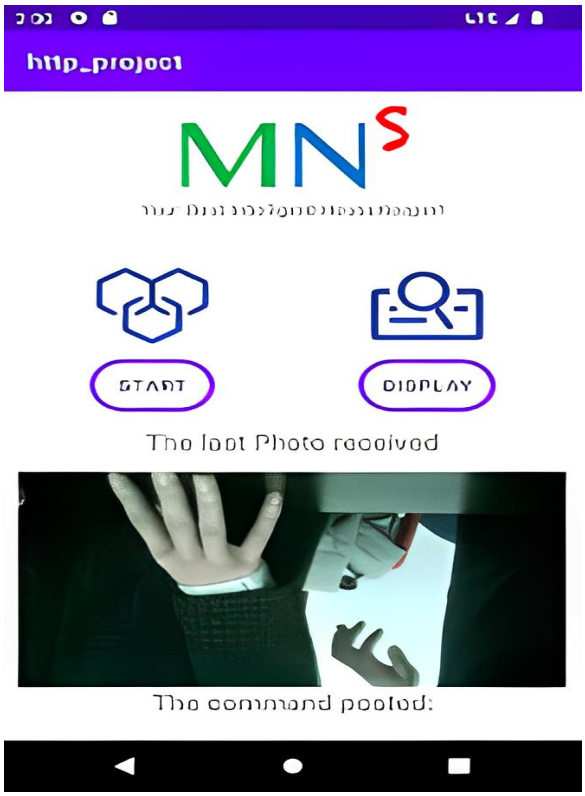

Figure 8. Diagram of the app UI.

4.5. Recognition Effect Display

According to Figure 9, the target recognition was carried out in a practical experimental operation. It is easy to see the high recognition accuracy and the ability to return commands and calculation results via the smartphone node.

The highest recognition accuracy of 99.82% was achieved at a camera resolution of 1028 × 1028 and when the area occupied by the recognized items was greater than 18.5%. All recognition results were essentially correct.

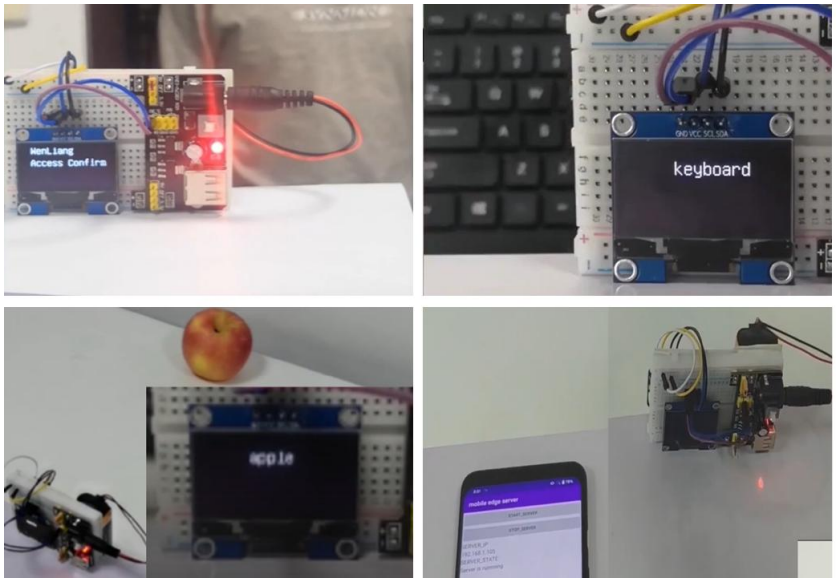

Figure 9. Experiment and recognition effect display.

5. Evaluation

According to the data given by Ali Dharma Institute and through relevant experimental attempts, we obtained the following data.

Table 1 shows that LACE reduces latency by 49.01% compared to traditional edge gateway services while lowering the cost to almost zero. Conservatively, it is estimated that LACE can reduce latency by at least 10% compared to traditional cloud computing services in multiple packet transmission.

Table 1. Comparison with types of service.

Method	Product	Core Device	Specification	Latency (5 KB)	Cost($/Month)
Cloud computing	Huawei cloud	General computing plus C6	8-cores 32 G	200 ms	171.74
Edge gateway	Yuanan IoT	COTX-SA	256-cores 32 GB	215 ms	>388.55
LACE	LACE	Huawei P30	8-cores 32 GB	70 ms	0

A test experiment was carried out on this intelligent access system at different distances between the front-end devices and edge computing servers, different movement speeds of edge computing servers, and different CPU loads. The average computation time of processing each video frame is the performance metric. The control variable method illustrates the average processing time by comparing the time from each video frame's sending to the result's reception in different conditions [28].

As shown in Table 2, it summarises the evaluation of the service performance of different mobile edge servers with varying distances between task offloading nodes and mobile edge servers, the CPU load of mobile edge servers, and movement, using the average processing time per frame as a test metric. As can be seen from the results, the average processing latency per frame tends to increase significantly with increasing CPU load for a given distance. The average processing latency per frame increases for a given CPU load as the distance increases. As the user moves faster, the average processing time also increases.

Table 2. Service performance at different distances, CPU load and movement speeds.

Distance (m)	CPU Load (%)	Movement Speed (km/h)	Average Computing Time (ms)
0	0	0	90
0	50	0	3985
0	100	0	4760
5	0	0	436
5	50	0	3228
5	100	0	5437
15	0	0	4275
15	50	0	5746
15	100	0	6690
<10	0	5	106
<10	100	5	4437
<10	0	10	123

Shortcomings and Future Directions for Improvement

In this experiment, we have only completed the basic effect test, and we have not performed a practical performance evaluation to illustrate the feasibility of our method for large-scale access to IoT devices, frequent access and contact with intelligent machines, the massive amount of data, and high accuracy required for calculation results. It also includes how to select offloading nodes for tasks to increase the likelihood of successful offloading.

In future work, we will optimize the proposed relevant communication protocols, improve the adaptive packet size partitioning algorithm, and provide better parameter optimization solutions. Similar to the greedy optimization algorithm mentioned in the literature [29], we will choose a more reasonable task offloading solution that may improve our offloading success rate.

6. Conclusions

This paper investigates the problem of task offloading in an intelligent building access control system. In the implementation, it can be seen that there is a high level of recognition accuracy and low latency. Compared to traditional cloud computing methods, not only is the feedback latency lower, but the reliability of the data is also ensured, avoiding problems such as errors caused by malicious interception or modification of data uploaded during internet distribution. Moreover, our proposed method's computational and spatial models have high accuracy. The proposed new task offloading scheme can effectively reduce the response time of IoT application devices and improve the efficiency of task offloading.

In future work, we will pay more attention to data security issues and implement some of the theoretical methods mentioned in this paper. The PID adaptation method is used to automatically adjust the size of the transmitted packets to reduce the cost of computational resources caused by repeated calculations in breakpoint sequencing. In addition, how to improve the accuracy of face recognition to ensure that faces can be correctly identified in the presence of occlusion.

Author Contributions: writing—original draft preparation, H.H. and H.T.; writing—review and editing, X.X.; methodology, J.Z.; supervision, Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (No. 61972074 and No. 61972075), the Natural Science Foundation of Sichuan Province (No. 2022NSFSC0885), and the National Key Research and Development Program of China (No. 2020YFE0200500).

Data Availability Statement: The data that has been used is confidential.

Conflicts of Interest: The authors declare no conflict to interest.

References

1. Belani, D.; Makwana, A.H.; Pitroda, J.; Vyas, C.M. Intelligent building new era of todays world. In Proceedings of the Trends and Challenges of Civil Engineering in Todays Transforming World, Surat, India, 17 March 2014.
2. Li, W.; Xue, K.; Xue, Y.; Hong, J. TMACS: A Robust and Verifiable Threshold Multi-Authority Access Control System in Public Cloud Storage. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 18. [CrossRef]
3. Mon, E.E.; Naing, T.T. The privacy-aware access control system using attribute-and role-based access control in private cloud. In Proceedings of the 2011 4th IEEE International Conference on Broadband Network and Multimedia Technology, Shenzhen, China, 28–30 October 2011; pp. 24–36.
4. Li, Q.; Ma, J.; Li, R.; Liu, X.; Xiong, J.; Chen, D. Secure, efficient and revocable multi-authority access control system in cloud storage. *Comput. Secur.* **2016**, *59*, 45–59. [CrossRef]
5. Loukatos, D.; Lygkoura, K.A.; Maraveas, C.; Arvanitis, K.G. Enriching IoT Modules with Edge AI Functionality to Detect Water Misuse Events in a Decentralized Manner. *Sensors* **2022**, *22*, 4874. [CrossRef] [PubMed]
6. Yang, S.U.; Shu, J.; Zhang, Z.; Software, S.O. Intelligent Access Control System based on Bluetooth Technology. *Intell. Comput. Appl.* **2014**, *2*, 9–17 .
7. Turner, V.; Reinsel, D.; Gantz, J.F.; Minton, S. The Digital Universe of Opportunities: Rich Data and Increasing Value of the Internet of Things. *J. Telecommun. Digit. Econ.* **2014**, *2*, 19–29.
8. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
9. Wang, R.; Tsai, W.T.; He, J.; Liu, C.; Li, Q.; Deng, E. A Video Surveillance System Based on Permissioned Blockchains and Edge Computing. In Proceedings of the 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), Kyoto, Japan, 27 February–22 March 2019; pp. 19–24.
10. Yu, Z.; Hu, J.; Min, G.; Zhao, Z.; Miao, W.; Hossain, M.S. Mobility-aware proactive edge caching for connected vehicles using federated learning. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 5341–5351. [CrossRef]
11. Cong, R.; Zhao, Z.; Min, G.; Feng, C.; Jiang, Y. EdgeGO: A mobile resource-sharing framework for 6g edge computing in massive IoT systems. *IEEE Internet Things J.* **2021**, *9* , 12–13. [CrossRef]
12. Soyata, T.; Muraleedharan, R.; Funai, C.; Kwon, M.; Heinzelman, W. Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In Proceedings of the Computers & Communications, Amsterdam, The Netherlands, 17 March 2012; p. 18.
13. Tang, K.S.; Man, K.F.; Chen, G.; Kwong, S. An optimal fuzzy PID controller. *IEEE Trans. Ind. Electron.* **2001**, *48*, 757–765. [CrossRef]
14. Zhou, G.; Birdwell, J.D. PID autotuner design using machine learning. In Proceedings of the Computer-Aided Control System Design, Napa, CA, USA, 17–19 March 1992; pp. 12–22.
15. Strm, K.J.; Hgglund, T. PID controllers: Theory, Design and Tuning. *Instrum. Soc. Am. Res. Triangle Park* **1995**, *23*, 26.
16. Ji-Ai, H.E.; Zheng-Hua, D.A. Simulation of BP Neural PID Controller. *J. Gansu Educ. Coll. Sci. Ed.* **2005**, *56*, 19–33.
17. Perkins, C.E.; Jagannadh, T. DHCP for mobile networking with TCP/IP. In Proceedings of the IEEE Symposium on Computers & Communications, Alexandria, Egypt, 27–29 July 1995; pp. 5–7.
18. Wang, J.; Hu, J.; Min, G.; Zomaya, A.Y.; Georgalas, N. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 242–253. [CrossRef]
19. Banegas, D.M.; Gracià, R.S. Offloading personal security applications to the Network Edge: A mobile user case scenario. In Proceedings of the Wireless Communications & Mobile Computing Conference, Paphos, Cyprus, 5–9 September 2016; pp. 22–40.
20. Kan, T.Y.; Yao, C.; Wei, H.Y. QoS-Aware Mobile Edge Computing System: Multi-Server Multi-User Scenario. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–16.
21. Ma, L.; Zhuo, Y.; Liao, K.; Liu, S.; Qiao, J.; Han, Z.; Wang, J. Development and research of digital campus system based on android. *Int. J. Smart Home* **2014**, *8*, 25–36. [CrossRef]
22. Wisanto, A.A.; Triwidiastuti, F.; Priyadi, P.R. Implementasi Sekuritas pada Web Service dengan Menggunakan Autentifikasi OkHttp pada Library Retrofit di Piranti Mobile. *JI-Tech* **2020**, *16*, 35–41.
23. Belkhir, A.; Abdellatif, M.; Tighilt, R.; Moha, N.; Guéhéneuc, Y.G.; Beaudry, É. An observational study on the state of REST API uses in android mobile applications. In Proceedings of the 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Montréal, ON, Canada, 25–26 May 2019; pp. 66–75.
24. Schroff, F.; Kalenichenko, D.; Philbin, J. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 815–823.
25. Ding, H.; Zhou, S.K.; Chellappa, R. FaceNet2ExpNet: Regularizing a Deep Face Recognition Net for Expression Recognition. In Proceedings of the 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), Washington, DC, USA, 30 May–3 June 2017; pp. 44–47.
26. Wu, Z.; Chen, X.; Gao, Y.; Li, Y. Rapid Target Detection in High Resolution Remote Sensing Images Using YOLO Model. *Isprs-Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2018**, *42*, 1915–1920. [CrossRef]
27. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.

28. Loukatos, D.; Arvanitis, K.G. Multi-Modal Sensor Nodes in Experimental Scalable Agricultural IoT Application Scenarios. In *IoT-Based Intelligent Modelling for Environmental and Ecological Engineering*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 101–128.
29. Hu, B.; Tian, X.; Yang, C.; Jiang, W.; Dong, S.; Chen, M.; Chen, W.; Chen, S.; Weng, Z. A dynamic resource chain task unloading method based on improved greedy algorithm. *J. Phys. Conf. Ser.* **2021**, *1883*, 23–25.