An IoT-Edge-Server System with BLE Mesh Network, LBPH, and Deep Metric Learning

Archit Gajjar, Shivang Dave, T. Andrew Yang, Lei Wu, and Xiaokun Yang

1 Introduction

In the last decade, the applications of embedded systems have boosted in the field of Internet-of-Things (IoT). Typically, IoT systems with multiple sensors including computation devices scattered over an enormous area [1]. While the advent of IoT solved many hitches, several inevitable problems were invited as well [2, 3]. The amalgamation of IoT and cloud requests facilitated the formation of edge computing, in which computing befalls at the network edge where there is no limitation of devices in terms of hardware type [4, 5]. In other words, the computing hardware device can be anything such as Raspberry Pi [6], Field Programmable Gate Array (FPGA) [7], System-on-Chip (SoC) [8], Application-Specific Integrated Circuit (ASIC) [9], general-purpose Central Processing Unit (CPU), or server [10]. For better understanding, a comparison between edge computing and traditional cloud computing systems is shown in Table 1.

Under the described idea, many research groups are working on edge computing in the hunt for further exploration and improvement. The edge computing is, currently, one of the most popular topics, and with the trend of machine learning, scholars are combining two ideas in order to achieve desired goals that are mentioned above [11, 14]. In the field of edge computing, the major focus of the research

A. Gajjar · S. Dave · T. Andrew Yang · X. Yang (✉)
Department of Engineering, University of Houston Clear Lake, Houston, TX, USA
e-mail: YangXia@UHCL.edu

T. Andrew Yang
Department of Computer Science, University of Houston Clear Lake, Houston, TX, USA

L. Wu
Computer Science Department, Auburn University at Montgomery, Montgomery, AL, USA

H. R. Arabnia et al. (eds.), *Advances in Artificial Intelligence and Applied Cognitive Computing*, Transactions on Computational Science and Computational Intelligence, https://doi.org/10.1007/978-3-030-70296-0_55

Table 1 Cloud computing vs. edge computing

Parameters	Cloud computing	Edge computing
Architecture	Centralized	Distributed
Data processing	Far from the source	Adjacent to the source
Latency	High	Low
Jittering	High	Low
Data privacy	Low	High
Accessibility	Global	Local
Mobility	Limited	Feasible
No. of nodes	Few	Extremely high
Data exploitation risk	Global	Remains in edge network
Communication with devices	Over the Internet	Local through edge node

work is on the software side, which includes performance improvement [19–21], algorithm optimization [12, 15], and increased efficiency with better task scheduling [13]. Comparatively, there is much less flow where scholars are working on the FPGA [17, 18], digital circuit [16], embedded systems [24], and hardware architecture [25, 26].

On that front, this literature proposes a promising architecture on IoT-Edge-Server based embedded systems with Bluetooth Low Energy (BLE) mesh system, surveillance system, a couple of processors and a server. In a generic manner, we have two platforms: (a) a BLE mesh network and (b) a surveillance demonstration. These systems are able to fetch data, at one instance, to Intel i7-7700HQ CPU and to Raspberry Pi on the other instance. The entire system has, virtually, three layers: (1) disposed layers, (2) edge computing network, and (3) cloud computing network [27].

The edge computing is boon to the IoT, but it also brings veil challenges that inspired us to propose a pioneering architecture. The main contributions of this work are as follows:

- To provide a robust architecture of IoT-Edge-Server embedded systems that can be implemented on extensively scattered environments such as agricultural farms, smart cities, or commercial/industrial buildings.
- Depending upon the application-specific tasks, we provide a system with an architecture that has two conceivable methods of computation: 1) traditional cloud computing and 2) edge computing.
- An implementation of two of the innumerable face recognition algorithms on Raspberry Pi and Intel i7-7700HQ CPU. There were few publications where they had performed a similar task on the comparably likewise hardware setting, but there is no mention of computation time or accuracy [28–30]. While our work not only delivers computation time and accuracy but also provides those data with an assortment of multiple face recognition algorithms, cameras, and computation devices as well.
- Improved data privacy—edge computing enables computation at the edge nodes, which requires personal data to be on the edge nodes rather than storing them on

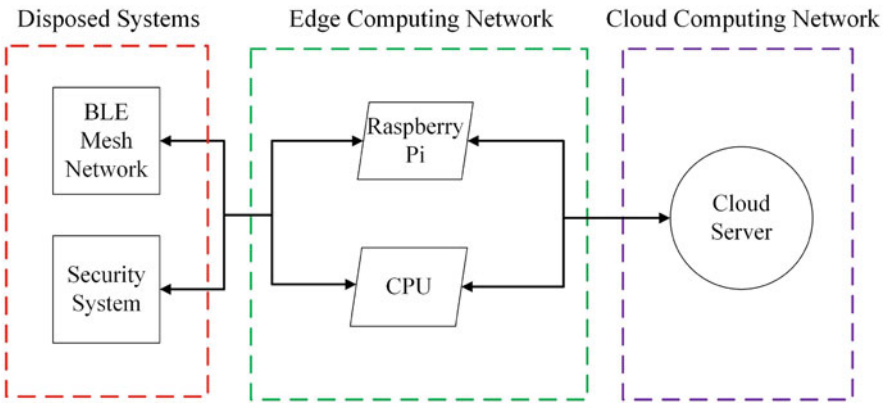

Fig. 1 Two channel computation options

a cloud. In other words, data remains at the network edge, which restricts data being hacked by anyone from the server.
- Improved computation speed—Performing any type of computation at any edge node prevents extreme back-and-forth of data to the cloud server and also improves the computation speed by reducing the latency. We provide benchmarks for computation time of cloud computing and edge computing for the same task to prove our hypothesis.
- We design communication commands for BLE mesh fabricated boards, which are reserved for only these boards.

2 Proposed Architecture

In this section, we propose the design architecture as shown in Fig. 2, and we also provide various computation options as shown in Fig. 1. The computation options can be decided based on the requirements in terms of computation time and responsiveness.

Figure 2 displays the proposed architecture of the IoT-Edge-Server based embedded system. More specifically, the architecture consists mainly of three virtual layers. These layers are depicted in Fig. 1: (1) disposed layer, (2) edge computing network, and (3) cloud computing.

2.1 Disposed System

In Fig. 2, the disposed system is labeled with the multiple disposed systems, which are BLE mesh network and surveillance system. Generally, the disposed layer is an integrated system to the edge computing network depending upon the applications

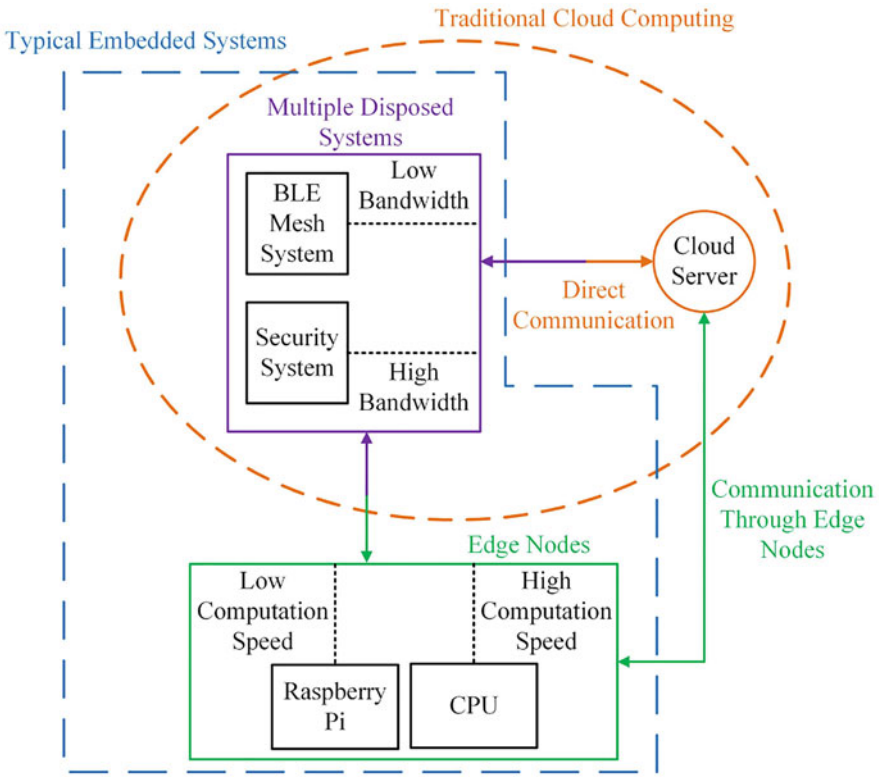

Fig. 2 Proposed architecture

and requirements. As mentioned earlier, for our proposed architecture and mainly to test our hypothesis, we implemented a BLE mesh network and surveillance system.

The BLE mesh network is a creation of four semi-customized boards using the APlix CSR 1020 modules that are suitable for low-power and restricted-complexity IoT applications. In this typical system, there is one master host connected to three slave boards. Each board possesses components to perform a designated task: (1) temperature and humidity sensor, (2) light-emitting diode (LED), and (3) motor.

On the other side, the surveillance system manifests a passive infrared (PIR) sensor associated with a camera to detect and recognize a face(s) in the milieu.

2.2 Edge Network

As shown in Fig. 2, the green-colored area named as edge nodes consists of all the computation devices for the architecture. Such devices are connected to the disposed systems and cloud server that apparently creates an edge computing network. For

time being, we neglect the orange-colored area to visualize a simple embedded system without any cloud server. For our system, we propose three types of edge nodes that include FPGA, CPU, and Raspberry Pi. These devices will provide assistance in the improvement of latency for the task and sieving data that needs to be stored on the server eventually. The filtration of data includes the removal of repetitive data, duplicated logs, or any junk data.

2.3 Cloud Network

The entire orange-colored system titled as traditional cloud computing, as displayed in Fig. 2, is envisioned as a cloud computing network where all the data from the disposed systems can be transferred to the server for the computation to take further steps. When the computation is completed, the decision or data will be sent back to the disposed systems. We would like to emphasize that the orange-colored portion in Fig. 2 is a traditional cloud computing architecture where all the computing and data would be sent to cloud for any kind of processing. For the proposed research work, the server specifications are as follows: Intel(R) Core (TM) i3-7350K @ 4.20 GHz with 8 Gigabyte (GB) Random Access Memory (RAM).

3 Implementation

In this section, details of the design of the whole platform are introduced including the BLE mesh system, surveillance system, and server-side execution.

3.1 BLE Mesh System

As mentioned before, the BLE mesh system has, currently, four self-designed boards accomplishing varied tasks creating the complete system on the foundation of the BLE low energy protocol. As shown in Fig. 3, using the mesh-topology method and abundant sensors, we can establish a large-scale environment. Figure 3 visually elucidates the arrangement of all the sensors in the giant setting. Moreover, the maximum distance between two boards to be able to communicate is around 57 feet as our experimental result. As shown in Fig. 3, the BLE mesh host/server is only able to communicate with an actuator that is in the same group 2. Furthermore, the actuator works as a relay for groups 1 and 2; in that manner, the BLE mesh host/server is able to communicate with the LED and temperature/humidity sensor boards. With the use of a relay between two boards, the maximum-communication distance can be increased to 77 feet as our results. Thus, a large network with numerous sensors can be deployed for wide-range habitats.

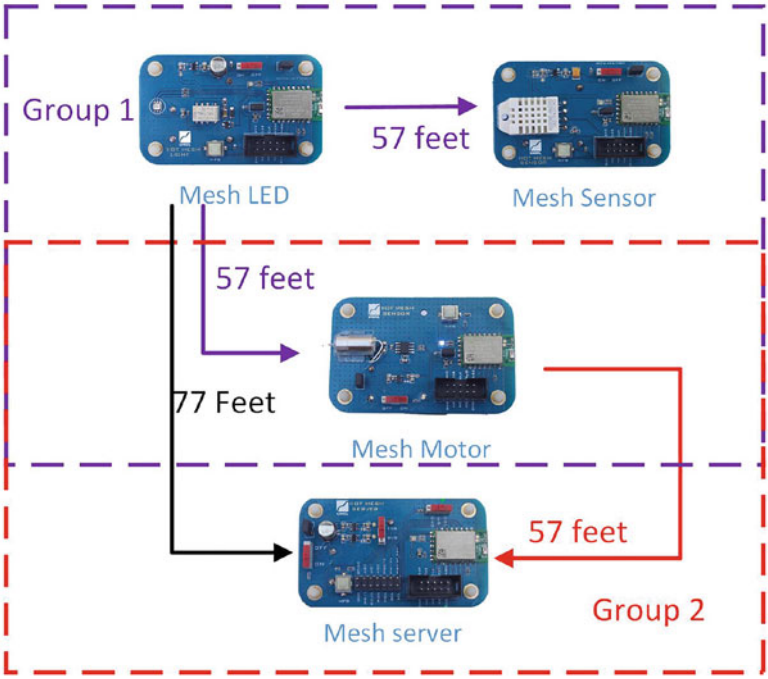

Fig. 3 Mesh topology

Data Type	Size	Content	Description
Header	2 B	0xFA, 0xF5	Data start header
Size	1 B	0x00 to 0x20	No more than 32
Data	N B	0xFF... 0xFF	Data frame
Checker	1 B	0x00 to 0xFF	Addition checking bit, including size and data
Stop	2 B	0x0D, 0x0A	Stop tail

Fig. 4 Data packet

The above-mentioned boards communicate on specific commands with each other. Due to the self-manufactured PCB boards, they are not, commercially, available in the market and also should interact with each other on the private set of commands that are reserved for these typical boards only [22, 23]. Figure 4 is a generic hexadecimal command, which includes required data types with their predefined size, content, and description for better understanding.

Figure 5 presents a particular hexadecimal command to turn on/off the LED.

Figure 6 demonstrates a particular hexadecimal command to control the LED, and the output is the mixed color of Red, Green, and Blue.

Header	Size	Type	On/Off	Checker	Stop
FA, F5	4	F2	00~01	XX	0D, 0A

Fig. 5 LED ON/OFF

Header	Size	Type	On/Off	Red	Green	Blue	Checker	Stop
FA, F5	7	F1	00~01	00~FF	00~FF	00~FF	XX	0D, 0A

Fig. 6 LED control

Header	Size	Type	Device No.	Checker	Stop
FA, F5	4	AB	02,03,04	XX	0D, 0A

Fig. 7 Mesh network check

Header	Size	Type	Direction	Strength	Checker	Stop
FA, F5	5	F4	00~01	00~05	XX	0D, 0A

Fig. 8 Motor control

Figure 7 is mainly used to check the network connection between the host and other devices. Note that device numbers hexadecimal 02, 03, and 04 allow to check network connection with LED, a thermal sensor, and motor, respectively.

Figure 8 displays the command to control a motor in which direction has two input values: hexadecimal 00 or 01 for clockwise and anti-clockwise rotation, respectively. The strength represents the speed of the motor that has a total of six input values: hexadecimal 00 stops the motor and hexadecimal 01 to 05 changes the speed of the motor from low to high.

Figure 9 provides ability to control the sensor. Here, hexadecimal 00 stops data receiving from the sensor and hexadecimal 01 activates data receiving, which provides current temperature and humidity.

The command, shown in Fig. 10, transmits current humidity and temperature value data to host that has 16-bit resolution for each, the most significant bit (MSB) to least significant bit (LSB). The sensor data is 10 times of the real humidity. On the other side, the MSB is the sign bit for temperature, where 1 and 0 represent negative and positive temperature, respectively. The other 15 bits are the value of temperature that is again 10 times of the real temperature.

– An example for humidity:
  Humidity range: hexadecimal 0000 to FFFF
  – 0000 0010 0101 0011 (Binary) = 0253 (Hex)
    0253 (Hex) = $2 \times 256 + 5 \times 16 + 3 = 595$
    Real humidity = 59.5% RH

Header	Size	Type	Sensing Data	Checker	Stop
FA, F5	4	F1	00~01	XX	0D, 0A

Fig. 9 Sensor control

Header	Size	Type	Humidity	Temperature	Checker	Stop
FA, F5	7	F3	0000~FFFF	0000~FFFF	XX	0D, 0A

Fig. 10 Sensor data

– An example for temperature:
  Temperature range: hexadecimal 0000 to FFFF
  – 0000 0001 0010 0001 (Binary) = 0121 (Hex)
    0121 (Hex) = $1 \times 256 + 2 \times 16 + 1 = 289$
    Real temperature = 28.9° C
  – 1000 0000 1000 0011 (Binary) = 8083 (Hex)
    8083 (Hex) = $0 \times 256 + 8 \times 16 + 3 = 131$
    Real temperature = -13.1° C

3.2 Surveillance System

When such a big system is deployed on the large environments, security becomes one of the main concerns [31–33]. Keeping security, a crucial factor of our system, we implemented an intelligent control system that can detect motion with a PIR sensor. Moreover, a detected motion enables the camera that captures an image on that instance of motion to detect and recognize a human face(s) in that typical image frame. If an unknown human is detected, the system creates an alert and notifies to the designated person via email. Upon the recognition of a known person from the database, there will be no alert generated.

There are plentiful algorithms available to recognize the face but for the sake of simplicity we chose two of them to test on our system: (1) the Low Binary Pattern Histogram (LBPH) and (2) the Deep Metric Learning (DML).

3.2.1 Low Binary Pattern Histogram

The LBPH is one of the algorithms open-sourced in the Open-CV library [34]. Post-implementation, we noticed that LBPH provides lesser accuracy compared to the Deep Metric Learning algorithm but at the same time it also has lower computation time for the equal setup. We implemented LBPH on Raspberry Pi 3 and Intel(R) Core (TM) i7-7700HQ CPU @ 2.80GHz with 16 Gigabyte (GB) Random Access

Memory (RAM). We also need to create and train the dataset in order to recognize the face(s) using the algorithm. Training of the dataset is performed every time changes occur in the dataset; otherwise, we do not need to train and directly run the algorithm to recognize a face(s).

Due to the lower computation time, it was feasible to contrivance LBPH on the Raspberry Pi. For the LBPH, the lighting of the surrounding, distance of humans from a camera, and many other perilous situations may affect the accuracy of recognizing a person from the given image frame.

3.2.2 Deep Metric Learning

DML is a face recognition algorithm from the dlib C++ library, which is an open-sourced collection of machine learning applications and algorithms [35]. The DML delivers surprisingly higher accuracy but at the same time drains out the computation power of a processor compared to LBPH. The requirement of the computation resource on DML is high; thus, as a case study the execution was successfully achieved on the Intel(R) Core (TM) i7-7700HQ CPU @ 2.80 GHz with 16 Gigabyte (GB) Random Access Memory (RAM).

As shown in Fig. 13, for DML based face recognition system, which identifies the human face if it is in the image frame, there is no need to train the dataset as the open-source library is pre-compiled with human facial features. Hence, it directly allows implementing inference on the desired platform.

Furthermore, on top of the face recognition systems, we have implemented an alert system that notifies the designated person if the recognized face(s) is not in the dataset. To accomplish such a task, we instigated a script using Simple Mail Transfer Protocol (SMTP) that enables us to send an alert in the email to the authorities to initiate safekeeping regarded action. The benefit of such notification is that the nominated person can choose to take contemplating safety action without physically being in the range of the system if needed.

3.2.3 Server-Side Execution

As shown in Fig. 2, we set traditional cloud computing architecture to measure the computation time for each task such as recognition time for both LBPH and DML algorithms as well as BLE mesh network commands. We create a server using Node.js, a JavaScript run-time environment to execute JavaScript, scripting on an Intel(R) Core (TM) i3-7350K @ 4.20 GHz with 8 Gigabyte (GB) Random Access Memory (RAM). In Fig. 11, steps to create a server on a processor are shown including all the project environments, which must be followed by "nodemon app" command where the "app" is our project name.

As represented in Fig. 12, whenever web request is triggered by task or user, an appropriate Node.js script runs to request data on the cloud from the disposed systems. In a follow-up after that processed request, data is transmitted toward

```
global.__basedir = __dirname;

const express = require('express');
const dotasksController = require('./controllers/dotasksController.js');

const app = express();

app.set('view engine','ejs');
app.use('/',express.static('./'));
app.use('/assets',express.static('./assets'));
app.use('/uploads',express.static('./uploads'));

dotasksController(app);

app.listen(2000);
console.log('Now listening on port 2000');
```

Fig. 11 Creating server on processor

cloud from the disposed systems. Depending upon the requirements, the cloud server executes a task or performs computation and sends back data to the disposed systems, if needed. For this work, the cloud server will be performing all the tasks or computations in a python scripting environment. Also, for any continuous task, such a process could have back-and-forth data transmission from cloud to a disposed system or vice versa.

To control the whole system, we create a simple web page to perform all the tasks using Hypertext Markup Language (HTML) and Node.js. A screenshot from demo web page is shown in Fig. 13.

Figure 14 is a screenshot from the web page after a task being executed on the BLE mesh system, and it also shows the time of execution for a task on cloud computing.

Figure 15 displays a screenshot of a web page when the database is stored on the cloud for LBPH algorithm based face recognition. The database stores 300 images each for every person.

As mentioned before, every time we change the database, the algorithm is required to train the new database, and for the status update, screenshot is shown in Fig. 16.

Finally, after receiving the database and training them, the algorithm would be able to recognize the face(s) from the image/video frame. Figure 17 shows a correctly recognized image together with a screenshot and execution time.

In Fig. 18, we share a screenshot of the face recognition system using the DML algorithm including the total computation time.

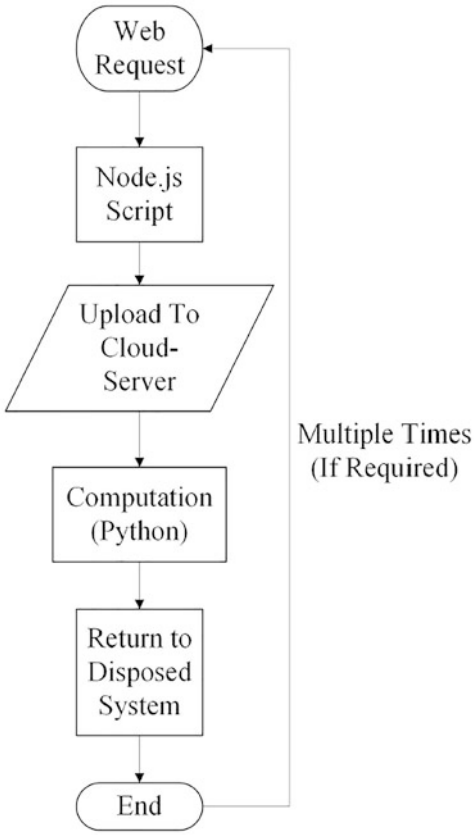

Fig. 12 Cloud server request flow

4 Experimental Results

In this section, we display results from the conducted experiments on the system to bolster our hypothesis. We share the distance improvement of the BLE mesh system with a node between them functioning as a relay. We provide accuracy results for the LBPH and DML algorithms on various computing devices including their performance results in terms of computation time.

Table 2 represents the maximum distance between two mesh boards, which is 57 feet. Interestingly, any board would not be able to communicate with other boards if the distance is greater than 57 feet, but it can be, definitely, increased up to 77 feet using other boards as a relay between those boards.

Table 3 represents computation time for a specific task on a couple of computation devices as well as a cloud server. While execution time on the Raspberry Pi and CPU for a specific same task is quite negligible, the execution time cloud server is terribly high even for such a low-data-rate task execution.

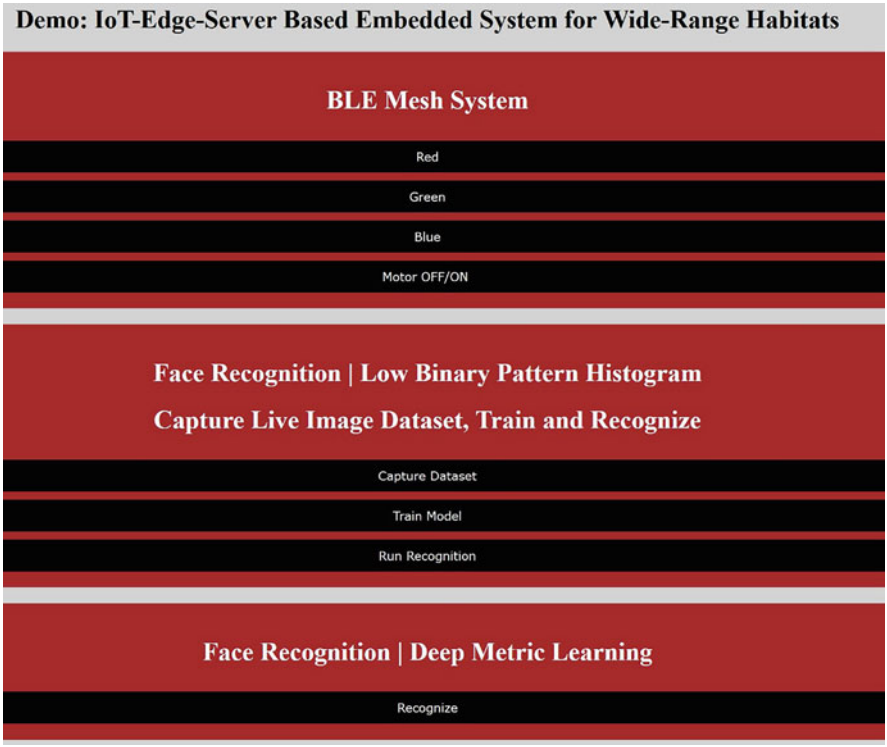

Fig. 13 Demo web page

Fig. 14 BLE mesh system—task execution

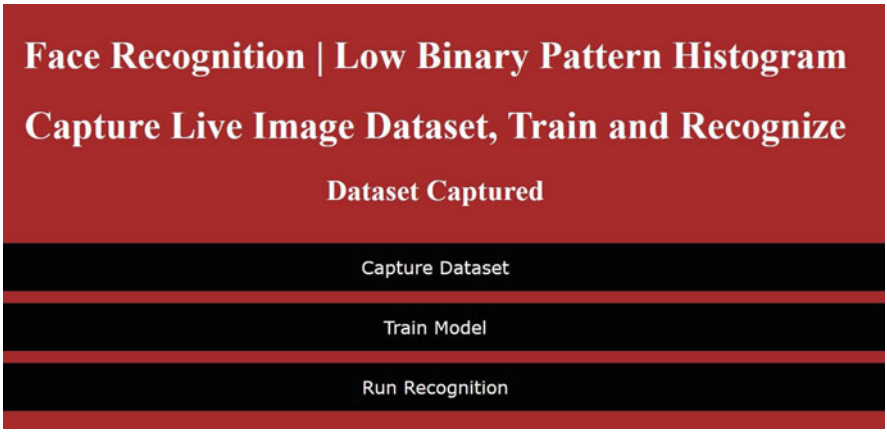

Fig. 15 Capture dataset for LBPH

Fig. 16 Trained dataset

Table 4 displays the time taken, in seconds, by different devices along with both the recognizing methods for the edge computing as well as execution for cloud computing. The computation time utterly relies on the hardware, and it is the processing speed. Furthermore, the time taken by the cloud server is fairly high compared to any edge computation devices.

Table 5 is the accuracy of recognizing the face(s) on the different systems with the same environment setting. The LBPH provides extremely lower accuracy compared to the DML method that consists of potential raise to a certain percentage by training more datasets and also replacing with better camera quality. Although there is a plausible way to increase accuracy, we exceedingly doubt about LBPH's capability to imitate the accuracy of DML.

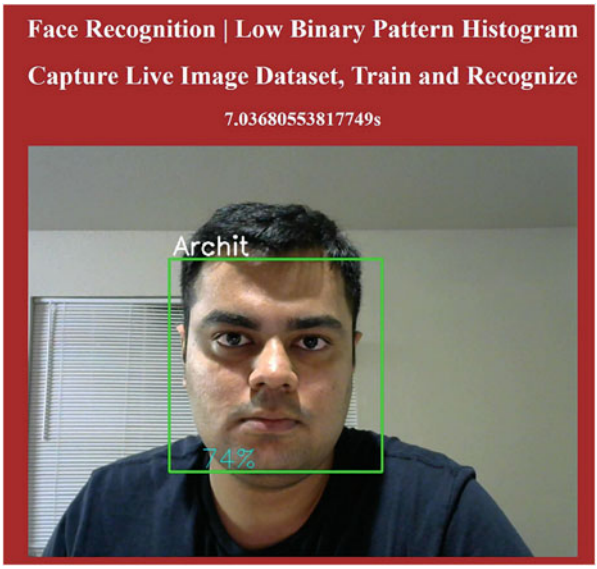

Fig. 17 Face recognition using LBPH

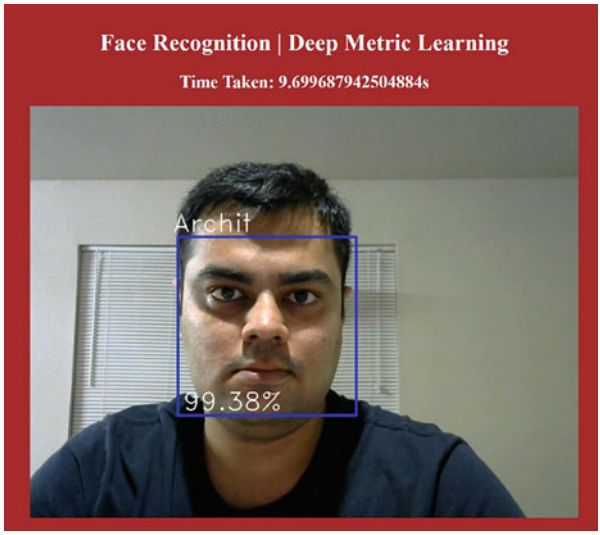

Fig. 18 Face recognition using DML

5 Conclusion

The proposed architecture presents favorable fallout as far as wide-range habitats are concerned with plausibility to deploy long-range BLE mesh system, scalable

Table 2 Distance results for mesh network

Maximum distance	No. of node(s)
57 feet	0
77 feet	1

Table 3 Task execution on BLE mesh system

Device	Task execution time(s)
Raspberry Pi	≥ 0.07
CPU	≥ 0.06
Cloud server	≥ 0.2

Table 4 Face recognition performance for edge/cloud computing

Device	Recognition time(s)	Method
Raspberry Pi	0.41–0.45	LBPH
CPU	0.035–0.037	LBPH
CPU	0.391–0.398	DML
Cloud server	≥ 6	LBPH
Cloud server	≥ 9	DML

Table 5 Face recognition accuracy

Devices	Recognition accuracy(%)	Webcam	Method
Raspberry Pi	50–60%	Logitech C270	LBPH
CPU	50–60%	Logitech C270	LBPH
CPU	60–70%	720p HD Laptop	LBPH
CPU	99.38%	720p HD Laptop	DML
Cloud server	50–60%	Logitech C270	LBPH
Cloud server	99.38%	Logitech C270	DML

surveillance system, and an email alert system. A large-scale BLE mesh system can be established using mesh topology. A scalable surveillance system means using a different combination of face recognition algorithms, camera quality, and hardware's computation speed, and one can use such a setup depending on the requirements. With the implementation of the proposed architecture, there are many application-specific large-scale sites that can be benefited such as commercial buildings, agricultural farms, industrial factories, etc.

6 Future Scope

The proposed architecture has many future research directions. The manufactured boards can be replaced with ASIC chips with research funding. As demonstrated in the proposed hardware architecture figure, all these computations can be done at a logic level using FPGAs that can be, eventually, replaced with ASIC chips, too. Alert system upgrades are also possible where mobile applications could be created rather than web page based control methods. Since the proposed work has a lot of

potential research opportunities in the emerging field of artificial intelligence, it is likely to create a big impact on the design of such an application-specific integrated circuit for offering low-latency and inexpensive computations.

References

1. P. Vangali, X. Yang, A compression algorithm design and simulation for processing large volumes of data from wireless sensor networks, vol. 7(4). *Communications on Applied Electronics (CAE)*, July 2017
2. X. Yang, L. Wu, et al., A vision of fog systems with integrating FPGAs and BLE mesh network. J. Commun. **14**(3), 210–215 (2019)
3. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
4. S.K. Datta, C. Bonnet, An edge computing architecture integrating virtual IoT devices, in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, pp. 1–3, Nagoya, 2017
5. H. El-Sayed et al., Edge of things: The big picture on the integration of edge, IoT and the cloud in a distributed computing environment. IEEE Access **6**, 1706–1717 (2018)
6. J. Thota, P. Vangali, X. Yang, Prototyping an autonomous eye-controlled system (AECS) using raspberry-pi on wheelchairs. Int. J. Comput. Appl. **158**(8), 1–7 (2017)
7. Y. Zhang, X. Yang, L. Wu, A. Gajjar, H. He, Hierarchical synthesis of approximate multiplier design for field-programmable gate arrays (FPGA)-CSRmesh system. Int. J. Comput. Appl. (IJCA) **180**(17), 1–7 (2018)
8. X. Yang, J. Andrian, A high performance on-chip bus (MSBUS) design and verification. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (TVLSI) **23**(7), 1350–1354 (2015)
9. X. Yang, J. Andrian, A low-cost and high-performance embedded system architecture and an evaluation methodology, in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, PP. 240–243, Tampa, FL, USA, 2014
10. X. Zhang, J. Lu, X. Fu, X. Yang, I. Unwala, T. Zhang, Tracking of targets in mobile robots based on camshift algorithm, in *Intl. Symposium on Measurement and Control in Robotics (ISMCR)* (UHCL, Houston, USA, 2019), pp. B2-3-1-B2-3-5
11. M. Alrowaily, Z. Lu, Secure edge computing in IoT systems: Review and case studies, in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 440–444, Seattle, WA, 2018
12. Y. Li, S. Wang, An Energy-aware edge server placement algorithm in mobile edge computing, in *2018 IEEE Intl. Conference on Edge Computing (EDGE)*, pp. 66–73, San Francisco, CA, 2018
13. P. Ren, X. Qiao, J. Chen, S. Dustdar, Mobile edge computing - a booster for the practical provisioning approach of web-based augmented reality, in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 349–350, Seattle, WA, 2018
14. J. Hochstetler, R. Padidela, Q. Chen, Q. Yang, S. Fu, Embedded deep learning for vehicular edge computing, in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, PP. 341–343, Seattle, WA, 2018
15. F. Wei, S. Chen, W. Zou, A greedy algorithm for task offloading in mobile edge computing system. China Communications **15**(11), 149–157 (2018)
16. X. Yang, Y. Zhang, L. Wu, A scalable image/video processing platform with open source design and verification environment, in *20th Intl. Symposium on Quality Electronic Design (ISQED 2019)*, pp. 110–116, Santa Clara, CA, US, April 2019
17. K. Vaca, A. Gajjar, X. Yang, Real-time automatic music transcription (AMT) with zync FPGA, in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, PP. 378–384, Miami, FL, US, 2019

18. Y. Zhang, X. Yang, L. Wu, J. Andrian, A case study on approximate FPGA design with an open-source image processing platform, in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 372–377, Miami, FL, US, 2019
19. A. Mebrek, L. Merghem-Boulahia, M. Esseghir, Efficient green solution for a balanced energy consumption and delay in the IoT-Fog-Cloud computing, in *2017 IEEE 16th Intl. Symposium on Network Computing and Applications (NCA)*, PP. 1–4, Cambridge, MA, 2017
20. S. Singh, Optimize cloud computations using edge computing, in *2017 Intl. Conference on Big Data, IoT and Data Science (BID)*, pp. 49–53, Pune, 2017
21. A. Yousefpour, G. Ishigaki, J.P. Jue, Fog computing: Towards minimizing delay in the internet of things, in *2017 IEEE Intl. Conference on Edge Computing (EDGE)*, pp. 17–24, Honolulu, HI, 2017
22. X. Yang, X. He, Establishing a BLE mesh network using fabricated CSRmesh devices, in *The 2nd ACM/IEEE Symposium on Edge Computing (SEC 2017)*, No. 34, Oct. 2017
23. A. Gajjar, Y. Zhang, X. Yang, A smart building system integrated with an edge computing algorithm and IoT mesh networks, in *The Second ACM/IEEE Symposium on Edge Computing (SEC 2017)*, Article No. 35, Oct. 2017
24. H. He, L. Wu, X. Yang, Y. Feng, Synthesize corpus for Chinese word segmentation, in *The 21st Intl. Conference on Artificial Intelligence (ICAI)*, pp. 129–134, Las Vegas, NV, USA, 2019
25. J. Grover, R.M. Garimella, Reliable and fault-tolerant IoT-edge architecture, in *2018 IEEE Sensors*, pp. 1–4, New Delhi, 2018
26. S.W. Kum, J. Moon, T. Lim, Design of fog computing based IoT application architecture, in *2017 IEEE 7th Intl. Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pp. 88–89, Berlin, 2017
27. A. Gajjar, X. Yang, H. Koc, et al., Mesh-IoT based system for large-scale environment, in *5th Annual Conf. on Computational Science & Computational Intelligence (CSCI)*, pp. 1019–1023, Las Vegas, NV, USA, 2018.
28. M. Sahani, C. Nanda, A.K. Sahu, B. Pattnaik, Web-based online embedded door access control and home security system based on face recognition, in *2015 Intl. Conference on Circuits, Power and Computing Technologies (ICCPCT)*, pp. 1–6, Nagercoil, 2015
29. I. Gupta, V. Patil, C. Kadam, S. Dumbre, Face detection and recognition using Raspberry Pi, in *2016 IEEE Intl. WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pp. 83–86, Pune, 2016
30. A. Gajjar, X. Yang, L. Wu, H. Koc, I. Unwala, Y. Zhang, Y. Feng, An FPGA synthesis of face detection algorithm using haar classifiers, in *Intl. Conference on Algorithms, Computing and Systems (ICACS)*, pp. 133–137, Beijing, China, 2018
31. X. Yang, W. Wen, Design of a pre-scheduled data bus (DBUS) for advanced encryption standard (AES) encrypted system-on-chips (SoCs), in *The 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 1–6, Chiba, Japan, Feb. 2017
32. X. Yang, W. Wen, M. Fan, Improving AES core performance via an advanced IBUS protocol. ACM J. Emerg. Technol. Comput. (JETC) **14**(1), 61–63 (2018)
33. X. Yang, J. Andrian, An advanced bus architecture for AES-encrypted high-performance embedded systems, US20170302438A1, Oct. 19, 2017
34. G. Bradski, The OpenCV library. J. Software Tools. https://docs.opencv.org/3.4.3/
35. D. King, Dlib-ml: A machine learning toolkit. J. Mach. Learn. Res. **10**, 1755–1758 (2009). http://dlib.net/