A Study of Integrated Server Platform for IoT Application Systems

September, 2024

Yohanes Yohanie Fridelin Panduman

Graduate School of
Natural Science and Technology

(Doctor's Course)
OKAYAMA UNIVERSITY

Dissertation submitted to
Graduate School of Natural Science and Technology
of
Okayama University
for
partial fulfillment of the requirements
for the degree of
Doctor of Philosophy.

Written under the supervision of

Professor Nobuo Funabiki

and co-supervised by
Professor Satoshi Denno
and
Professor Yasuyuki Nogami

Okayama University, September 2024.

To Whom It May Concern

We hereby certify that this is a typical copy of the original doctor thesis of
Yohanes Yohanie Fridelin Panduman

Signature of	Seal of
the Supervisor	
	Graduate School of
Prof. Nobuo Funabiki	Natural Science and Technology

Abstract

Nowadays, the *Internet of Things (IoT)* has attracted significant interest from both industrial and academic communities. The rapid developments of IoT technologies have increased the possibilities of realizing *smart cities*, *smart homes*, and *smart factories* where collaborations and integrations of various IoT application systems are essential.

However, IoT application systems have often been designed and deployed independently without considering the standards of devices, logics, and data communications. As a result, IoT application system developers need to design and implement the system by themselves, considering the standards for heterogeneous device managements and interoperability with other systems.

In this thesis, I present the study of an integrated IoT server platform called *Smart Environmental Monitoring and Analytical in Real-Time (SEMAR)* to support various IoT application systems using different standards. *SEMAR* provides the standard features for collecting, displaying, processing, and analyzing sensor data from various IoT devices. It can dynamically collect data from devices, process them, save them in the *Big Data* storage, and show them to users through a web-based user interface.

As the first contribution of the thesis, I present the design and implementation of the standard functions in *SEMAR*. It offers *Big Data* environments with rich *built-in* functions for data aggregations, synchronizations, filtering, and classifications with machine learning techniques. Besides, *plug-in* functions can be easily implemented and added there without modifying the existing codes. Data from devices for different sensors can be accepted directly or through network connections, which will be used in real-time for user interfaces. Data can be exported in text files, and be accessed from other systems through *Representational State Transfer Application Programming Interface (REST API)* services. It utilizes *Message Queue Telemetry Transport (MQTT)* and *REST API* for the communication protocol services.

For evaluations of the first contribution, the *SEMAR* platform is implemented and integrated with five IoT application systems. They include the *air-conditioning guidance system*, the *fingerprint-based indoor localization system*, the *water quality monitoring system*, the *environment monitoring system*, and the *air quality monitoring system*. When compared with existing researches on IoT platforms, the proposed *SEMAR* IoT application server platform offers higher flexibility and interoperability with the functions for IoT device managements, data communications, decision making, synchronizations, and filtering that can be easily integrated with external programs or IoT applications without changing the codes. The results confirm the effectiveness and efficiency of the proposed system.

As the second contribution of the thesis, I implement the *edge device framework* for *SEMAR* to remotely optimize the edge device utilization. It functions in three phases. In the *initialization phase*, it automatically downloads the configuration file to the device through *HTTP* communications. In the *service phase*, it converts data from various sensors into the standard data format and sends it to the server periodically. In the *update phase*, it remotely updates the configuration through *MQTT* communications. The edge configuration file includes the connectivity of sensor interfaces, a data conversion approach, a data model, transmitted data, local data storage, local visualization, and the data transmission interval on the server.

Besides, I implement filtering functions using digital signal processing techniques in this framework. The techniques include low, high, and band-pass filters based on *Butterworth* and *Chebyshev-I*, as well as *Kalman* and *Savitzky-Golay* filters. It also provides the cascading filter to create a series of filters for data processing in sequences. By identifying these techniques and parameters in the edge configuration file through *SEMAR*, the user can utilize them without writing codes.

For evaluations, I apply the proposal to the *data logging system* and the *fingerprint-based indoor localization system (FILS15.4)*. These integrated systems were deployed in #1 and #2 Engineering Buildings at Okayama University, Japan. In addition, I evaluate the effectiveness of the edge device framework by investigating its computing performance and comparing it with similar research works. The results confirm the effectiveness of utilizing *SEMAR* to develop IoT application systems.

As the last contribution of the thesis, I study the integration of *Artificial Intelligence (AI)* functions into *SEMAR*. Recently, AI has become very popular and widely used in various applications, including IoT. To support this growth, the integration of AI into *SEMAR* is essential to enhance its capabilities after identifying the current trends of applicable AI technologies in IoT applications. First, I provide a comprehensive review of IoT applications using AI techniques in the literature. They cover predictive analytics, image classification, object detection, text spotting, auditory perception, *Natural Language Processing (NLP)*, and collaborative AI. Second, I identify the characteristics of each technique by considering the key parameters, such as software requirements, input/output (I/O) data types, processing methods, and computations. Third, I design the integration of AI techniques into *SEMAR* based on the findings. Finally, I discuss the use cases of *SEMAR* for IoT applications with AI techniques.

In future works, I will continue to implement AI technologies in *SEMAR* to complete the proposed design. I will implement feedback functions with the PID control and the sequence control within the edge computing framework in *SEMAR* to enhance the functionality of the system for *Industry 4.0*.

Acknowledgements

This dissertation is only possible with an enormous amount of support I received during this study. Here, I would like to pay my highest appreciation to God and the individuals who have supported me by giving their valuable time and effort. I am nothing but grateful to receive such blessings.

First, I owe a debt of gratitude to my honourable supervisor, Professor Nobuo Funabiki. His excellent supervision, brilliant suggestions, and rigorous encouragements were not only helping my PhD research but also made me grow as a person. Only with his strongest support, I was able to motivate myself to learn a lot of new things, focus on my goal, and achieve it. I greatly admire his dedication to creating a positive study environment and his commitment to guiding his students. I look up to him as a teacher in many ways, and he is one of the most significant figures in my life. I am truly grateful for the opportunity to pursue my Ph.D. under his supervision.

I am indebted to my PhD co-supervisors, Professor Satoshi Denno and Professor Yasuyuki Nogami, for taking their valuable time to give me advice, guidance, insightful comments, and proofreading of this thesis.

I would like to express my gratitude to Dr. Sritrusta Sukaridhoto, who belongs to Politeknik Elektronika Negeri Surabaya, Indonesia, for his continuous support throughout my studies. His valuable advice and insightful discussions were immensely valuable to this thesis.

I extend my sincere thanks to Professor Minoru Kuribayashi and Professor Mitsuhiro Okayasu for their valuable support and discussions during this work.

Then, my great appreciation is also directed to all the members of the Distributed System Design Laboratory. Mrs. Keiko Kawabata and Ms. Safira Kinari for the administrative support; Dr. Pradini Puspitaningayu, Mr. Masaki Sakagami, Mr. Kazushi Hamazaki, Mr. Sho Ito, and Mr. Sakamaki Shunya who often worked together with me; Dr. Sujan Roy, Dr. Yuanzhi Huo, Ms. Irin Anggraini, Dr. Rahman Patta, Dr. Kwenga Ismael, and Dr. Hein Htet who helped me a lot, especially in the beginning of my study; Ms. Khaing Hsu Wai, Mr. Xudong Zhou, and Ms. Shune Lae Aung who have shared strength during our three years; Mrs. Evianita Dewi, Mr. Akai Leon, and all lab members in general. Though it is difficult to mention every name, none of the enjoyable moments we shared would ever slip from my memory.

Over and above, I thank all of my friends who shared fate with me as students in Okayama, away from our far home in Indonesia. Having emotional support from you gave me strength to a level I never knew I could have. It was truly a blessing to know all of you.

I would like to acknowledge the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT) for financially supporting my PhD study at Okayama University.

Last but not least, my utmost gratitude to my family and my girlfriend who gave me their unending and unconditional love. You are the reason I am here, and the reason I am coming home.

Yohanes Yohanie Fridelin Panduman
Okayama, Japan
September 2024

List of Publications

Journal Papers

1. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Pradini Puspitaningayu, Minoru Kuribayashi, Sritrusta Sukaridhoto, Wen-Chung Kao, "Design and implementation of SEMAR IoT server platform with applications," Sensors, vol. 22, no. 17, pp. 6436, 2022.

2. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Sho Ito, Radhiatul Husna, Minoru Kuribayashi, Mitsuhiro Okayasu, Junya Shimazu, Sritrusta Sukaridhoto, "An Edge Device Framework in SEMAR IoT Application Server Platform," Information, vol. 14, no. 6, pp. 312, 2023.

3. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Evianita Dewi Fajrianti, Shihao Fang, Sritrusta Sukaridhoto, "A Survey of AI Techniques in IoT Applications with Use Case Investigations in the Smart Environmental Monitoring and Analytics in Real-Time IoT Platform,"" Information, vol. 15, no. 3, pp. 153, 2024.

International Conference Papers

4. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Pradini Puspitaningayu, Masaki Sakagami, Sritrusta Sukaridhoto, "Implementations of integration functions in IoT application server platform," 5th International Conference on Vocational Education and Electrical Engineering (ICVEE 2022), pp. 72-77(Online, Surabaya, Indonesia, 2022).

5. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Sritrusta Sukaridhoto, "An Idea of Drone-Based Building Crack Detection System in SEMAR IoT Server Platform," In 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE), pp. 12-13 (Nara, Japan, 2023).

6. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Sritrusta Sukaridhoto, "Implementation of Digital Filter Functions in Edge Device Framework for IoT Application System," 6th International Conference on Vocational Education and Electrical Engineering (ICVEE 2023), pp. 274-279 (Online, Surabaya, Indonesia, 2023).

Other Papers

7. **Yohanes Yohanie Fridelin Panduman**, Nobuo Funabiki, Radhiatul Husna, Sritrusta Sukaridhoto, Wen-Chung Kao, “An Overview of Edge Device Framework in SEMAR IoT Application Server Platform,” IEICE Society Conference, pp. S11–12, (Saitama, Japan, 2023).

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

Nowadays, the *Internet of Things (IoT)* is receiving strong attentions from both industries and academics as an emerging technology that uses the Internet infrastructure to connect physical worlds to cyberspaces [1]. The IoT application infrastructure has continuously been extended to become more ubiquitous around the world. It is composed of numerous physical devices distributed across multiple domains [2]. In this context, IoT applications must provide interoperability functions to collect, store, and disseminate data from several sensors, and provide them to other systems [3, 4].

The rapid developments of IoT technologies have increased the possibilities of realizing *smart cities*, *smart homes*, and *smart factories*, where collaborations and integrations of various IoT application systems are essential. For the purposes, numerous studies on IoT server platforms have emerged. Without focusing on the implementation details, the prepared tools in the platform allow developers to focus on the implementation of logic by using functions that efficiently support the design and implementation of IoT applications [5]. However, several challenges appear in the effective management and analysis of IoT data through these platforms.

The first challenge involves the lack of standards for data sensors, processing, and communication protocols. An IoT server platform should be able to handle various data types from different sensors, which makes it necessary to be able to work with each other despite the diversity of communication protocols or sensor technologies.

The second challenge concerns the data interoperability between various IoT application systems. It can be described as the integration of plural systems by sharing output data through information networks [3]. The IoT server platform collects and processes data from a lot of devices and provides it to other systems [4]. As a result, IoT application system developers need to design and implement the system by themselves, considering the standards for heterogeneous device managements and interoperability with other systems.

The third challenge relates to the implementation of *edge computing* in an IoT application system. Edge computing brings computing capabilities for data processing to locations closer to sensors or target devices [6]. Some IoT applications may require low latency and real-time data processing, which cannot be provided by cloud servers [7, 8]. Due to the diversity of sensor resources, the introduction of *edge computing devices* has become a valuable solution to reducing the computational complexity of data processing in cloud servers [9]. Edge computing devices enable various functions at the edges of networks before sending data to the server and can increase the efficiency of data processing [10]. It also offers the data conversion capability to convert raw data to the standard data format. It is expected that the *edge device framework* was introduced to facilitate application developments in edge computing devices [11]. The framework interacts with devices in the physical world that may change over time [12]. Therefore, it should support the dynamic development of edge systems.

The fourth challenge involves the integration of *Artificial Intelligence (AI)* that will play critical roles in the evolution of IoT technology. Recently, AI has become very popular as a data processing algorithm. AI has been inspired by the thinking of the human brain [13]. It can create intelligent systems that may learn, operate, and respond intelligently like human behaviors [14, 15]. In the context of IoT, it enables advanced sensor data analysis by identifying data patterns, extracting valuable information, and making rapid decisions based on it [16]. Furthermore, the utilization of *Big Data* technologies enhances this integration by providing huge data sets for training AI models. This significantly increases the potential of AI implementations in various IoT applications. To support this growth, the integration of AI into an IoT application server platform becomes essential to enhance its capabilities.

1.2 Contributions

This thesis presents the proposal of an integrated IoT application server platform called *Smart Environmental Monitoring and Analytical in Real-Time (SEMAR)*. *SEMAR* can be a cloud server for integrating various IoT application systems using different standards. It provides the standard features for collecting, displaying, processing, and analyzing sensor data from various IoT devices on a single platform [17]. It can dynamically collect data from devices, process them, save them in the *Big Data* storage, and show them to users through a web-based user interface [18].

1.2.1 Implementation of Integration Functions in *SEMAR*

The first contribution of the thesis is the development of the standard functions in SEMAR [17,18]. It offers *Big Data* environments with rich *built-in* functions for data aggregations, synchronizations, filtering, and classifications with machine learning techniques. Besides, *plug-in* functions can be easily implemented and added there without modifying the existing codes. Data from devices for different sensors can be accepted directly or through network connections, which will be used in real-time for user interfaces. Data can be exported in text files, and be accessed from other systems through *Representational State Transfer Application Programming Interface (REST API)* services. It utilizes *Message Queue Telemetry Transport (MQTT)* [19] and *REST API* for the communication protocol services.

The functions were evaluated through integration with five IoT application systems. They include the *air-conditioning guidance system*, the *fingerprint-based indoor localization system*, the *water quality monitoring system*, the *environment monitoring system*, and the *air quality monitoring system*. The results confirm the effectiveness and efficiency of the proposed system, including the data transmission performance with the implemented *MQTT*.

1.2.2 Implementation of Edge Device Framework in *SEMAR*

The second contribution of the thesis is the implementation of the *edge device framework* for *SEMAR* to remotely optimize the edge device utilization [20, 21]. This framework allows users to remotely configure the connectivity of sensor interfaces, a data conversion approach, a data model, transmitted data, local data storage, local visualization, and the data transmission interval on the server. It functions in three phases. In the *initialization phase*, it automatically downloads the configuration file to the device through *HTTP* communications. In the *service phase*, it converts data from various sensors into the standard data format and sends it to the server periodically. In the *update phase*, it remotely updates the configuration through *MQTT* communications. As a popular edge device, the *Raspberry Pi* was selected for this implementation, and the image was created in the *SEMAR* server.

Besides, the thesis implements filtering functions using digital signal processing techniques in this framework [22]. The techniques include low, high, and band-pass filters based on *Butterworth* and *Chebyshev-I*, as well as *Kalman* and *Savitzky-Golay* filters. It also provides the cascading filter to create a series of filters for data processing in sequences. By identifying these techniques and parameters in the edge configuration file through *SEMAR*, the user can utilize them without writing codes.

The framework was evaluated by applying it to the *FILS15.4* and the *data logging system*. These integrated systems were deployed in #1 and #2 Engineering Buildings at Okayama University, Japan. In addition, the effectiveness of the edge device framework was evaluated by investigating its computing performance and comparing it with similar research works. The results confirm the effectiveness of utilizing *SEMAR* to develop IoT application systems.

1.2.3 Study of AI Techniques Integration with Use Cases in *SEMAR*

The last contribution of the thesis is the AI techniques integration into *SEMAR* [23]. It presents an overview of current AI techniques and their use cases in IoT applications. The proposed methodology explores the potential of AI integrations and how they can be implemented in IoT applications. First, a comprehensive review is provided on current studies on IoT applications using AI techniques. They include predictive analytics, image classification, object recognition, text spotting, auditory perception, *NLP*, and collaborative AI. Then, the characteristics of each technique are identified by considering the key parameters that play a critical role in integrations. These parameters include software requirements, *input/output (I/O)* data types, processing methods, and computations. Based on these findings, the seamless integration of AI capabilities is designed into the *SEMAR* platform. Finally, use cases of IoT applications with AI techniques are discussed to illustrate how *SEMAR* can be used to support their developments. Through three IoT application use cases, I illustrate how our designed platform supports and enhances IoT application development with AI processes.

1.3 Contents of This Dissertation

The remaining part of this thesis is organized as follows. Chapter 2 describes the architecture of the IoT application system. Chapter 3 presents the design and implementation of the integration functions in *SEMAR*. Chapter 4 presents the implementation of the *edge device framework* in *SEMAR*. Chapter 5 presents a comprehensive literature review of AI techniques in IoT applications with their use cases for designing AI techniques integrations in *SEMAR*. Chapter 6 reviews relevant works in literature. Finally, Chapter 7 concludes this thesis with some future works.

Chapter 2

Review of IoT Application System Architecture

This chapter introduces a review of the IoT application system architecture that is adopted for designing the IoT application server platform and the edge computing framework in this study.

2.1 Overview

This chapter describes the design of the IoT application system architecture for generalization. Currently, there are a lot of IoT architecture references that can be considered for IoT application systems. Each IoT application system may have unique designs and requirements, where the common IoT application system architecture can consist of three layers, as shown in Figure 2.1.

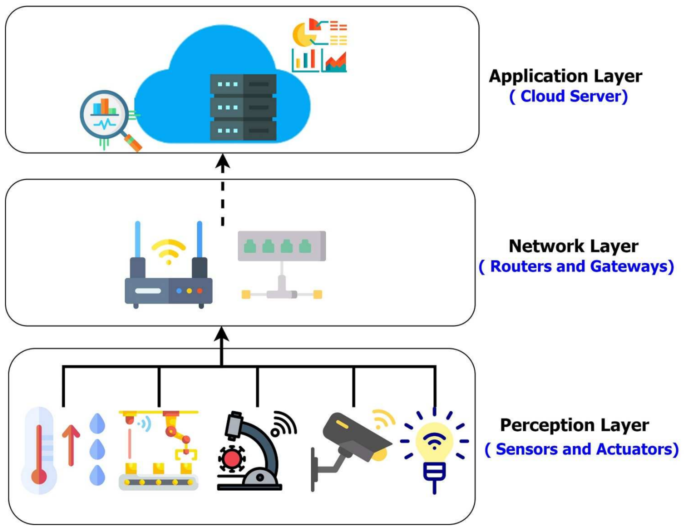


Figure 2.1: Three-layer IoT architecture

The *perception layer* represents the physical devices for sensing and actuating that interact with the environments, consisting of sensors and actuators. The *network layer* represents the transport layer for data communications between layers. The *application layer* represents the application software to offer specific services for data processing [24]. A cloud server is often used for this layer. There are various IoT application system architectures that need to be addressed to enhance the development of IoT applications and platforms.

In [25], Lombardi et al. presented commonly used IoT architectures such as *cloud-based* architecture, *edge-computing-based* architecture, and *Social Internet of Things (SIoT)* architecture. *Cloud-based* architecture utilizes services deployed on a cloud server to generate, process, and visualize large amounts of data for users. This architecture allows users and other services to access data at any time. *Edge-computing-based* architecture offers computational services close to the device layer by offering data processing, storage, and control capabilities. It is frequently used for industrial devices and IoT application systems that demand a quick response as a result of data processing.

In *SIoT* architecture, IoT applications are comprised of objects registered on a social networking platform, where each object collaborates and interacts with other objects to provide specific services [26]. This architecture enables IoT objects to conduct high-computational processes, as opposed to only the server performing these tasks. It enables the development of IoT applications that interact with one another. In addition, the *MIoT* architecture has been added to the *SIoT* architecture. In order to reduce the complexity of the *SIoT* architecture system, the *MIoT* architecture considers data-driven and semantics-based aspects for data exchange between objects [27].

In this study, the proposed concept of the IoT application system architecture is based on these references. Figure 2.2 illustrates the IoT architecture to be considered. It is composed of the *sensors and actuators*, *edge*, and *cloud* layers.

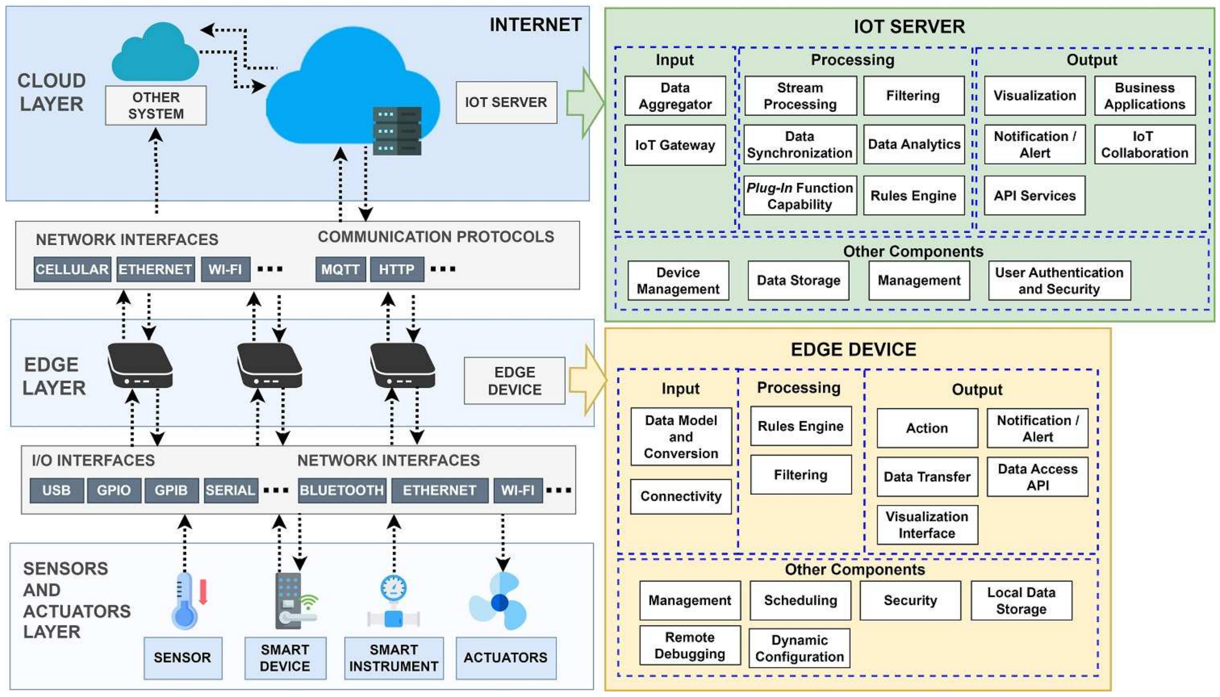


Figure 2.2: Design overview of general IoT application system architecture.

2.2 Sensor and Actuator Layer

In the context of the IoT application system, perception devices as IoT objects are sensors and actuators connected to a controller. Sensors are primarily used to monitor the environment by converting physical parameters into measurable electrical quantities (often voltage), while actuators provide physical actions when presented with electrical quantities. Besides, with the rapid development of technologies, Internet-connected devices have become common in their application purposes.

For instance, in smart homes, developers have often utilized smart devices to improve living experiences and reduce energy consumption. These smart devices are controlled by smartphones and are integrated with cloud services through wireless networks.

The *Industrial Internet of Things (IIoT)* has been presented to connect IoT technologies to industrial machines or instruments to analyze the obtained data and optimize existing industrial processes [28]. It uses smart instrument devices for automatic data collection to enhance the condition monitoring of industrial instruments. Recently, industrial devices in the market have contained features to enable Internet-based data access to central operation management systems through Ethernet and wireless technology. In this study, I consider smart devices and smart instruments as components in the sensor and actuator layers of the proposed architecture.

2.3 Edge Layer

The *edge* layer addresses the issue of the growing data volume in an IoT application system by utilizing the computing capabilities of edge devices. In this study, I explain the components of the edge device in *input*, *processing*, *output*, and *other* components, as illustrated in Figure 2.2.

The *input* components are responsible for collecting data from various sensors and devices. Since sensor devices usually generate data in different and non-standard formats, the edge device requires the data conversion component to generate data in the standard format from various sensor devices into valuable data structures. A data model is required for this purpose. The connectivity component refers to the *Input/Output (I/O)* and network interfaces of the IoT device for data communications. Currently, a single board computer such as the *Raspberry Pi* has several interfaces to accept data from a variety of devices. These are *General Purpose Input Output (GPIO)*, serial communications, Bluetooth, and Wi-Fi.

The *processing* components in the edge layer are designed to optimize data collections, and enable immediate analysis and decision-making. As an extension of cloud services, edge computing is able to perform local data processing with minimal computational resources. The filtering and the rules engine are included in these components. The *filtering* function reduces noises before transmitting data to a cloud server. The *rules engine* makes data-driven decisions in real-time.

The output components concern the ability of edge devices to utilize the collected data and transmit it to the cloud server or other systems. Several output components, such as the visualization interface, notification/alert, data transfer, trigger action operations, and data access API, should be considered for this purpose. To support the current IoT trends of cross-vendor capabilities and interoperability, the *data access API* brings edge devices to allow external systems to access local data through *HTTP* communications.

Network interfaces of the edge device and communication protocols need to be considered for data transmissions. The edge device, such as *Raspberry PI*, has allowed diverse network interfaces. Wi-Fi, Ethernet, and 5G cellular are standard network interfaces used to connect edge devices to cloud servers. Communication protocol services consist of the *publish-subscribe* and *request-response* messaging models. In addition, the standardization format of data transfer should be addressed for this component. In this case, the JSON format is utilized.

For developing the edge device, we should consider additional components that are not included in the *input*, *processing*, and *output* components. These components are management, scheduling, security, local data storage, remote debugging, and dynamic configuration. The *management* component controls and monitors the lifecycle of the edge device. As sensor data may not be always transmitted to the server due to network issues, the *local* data storage component archives sensor data records inside the edge device.

2.4 Cloud Layer

The cloud layer components are responsible for processing, analyzing, managing, storing, and visualizing IoT data using cloud-based services. These components perform computations that are not feasible on edge devices. In this study, I present the cloud layer components in Figure 2.2. I divide them into *input*, *processing*, *output*, and *other* components.

The *input* components provide the services to receive sensor data from different devices using different communication protocols. It consists of the IoT gateway and the data aggregator. The *IoT gateway* component in a cloud service should implement standard IoT data transfer communication protocols, such as *HTTP* and *MQTT*. The *data aggregation* component processes the data in a usable format. Each IoT gateway component may have different data aggregator services.

The *processing* components contain a variety of data processing functions for IoT data stream processing, filtering, rules engine, data synchronizations, and analytics, where each function should be implemented as a standalone one to prevent system failures. By incorporating *plug-in* functions, IoT servers can gain more valuable functionality for data processing. It enables diverse IoT applications to address unique use cases and requires specific data processing that has not yet been implemented into the system.

The *output* part concerns the ability of the cloud system to provide capabilities for users or other systems to access IoT data. The output components may include visualization functions, notification/alert functions, *REST API* services, business application integrations, and IoT collaboration capabilities. The *other* components provide additional components that will support the main services of the cloud server. They include management, data storage, device management, user authentication, and security components.

The data process at the cloud layer usually starts when the *IoT gateway* receives sensor data. It will be followed by *data aggregation*. The *data aggregation* component collects data from several data sources, applies data processing, and reassembles data in a usable format. Then, data will be forwarded to the *processing* components and be stored in the data storage. The processing components allow extracting the necessary information from the collected data. Finally, the output components provide the result through the *visualization* and the *notification/alert* functions. In addition, the *REST API* services allow for other systems to access IoT data to support system collaboration and interoperability.

2.5 Summary

This chapter provided an overview of the IoT application system architecture, highlighting the important features of both the edge device and cloud layers. It presented the roles of *input*, *processing*, *output*, and *other* components, as well as the importance of network interfaces and communication protocols for connecting the edge device and cloud layers. The IoT application server platform and edge device framework in this study are designed by considering this architecture.

Chapter 3

Design and Implementation of Integration Functions in SEMAR

This chapter presents the design of the *SEMAR* IoT server platform for integrating various IoT application systems. It describes the implementation of integration functions for communication, collection, display, processing, and analysis of sensor data.

3.1 System Overview

Figure 3.1 shows the proposed design of the *SEMAR* IoT server platform. The main components are *data input*, *data processing*, and *data output*. The *data input* is responsible for accepting data from various sources. The *data process* provides the modules for data processing, control, and collection. The *data output* enables visualizations and sharing of collected data.

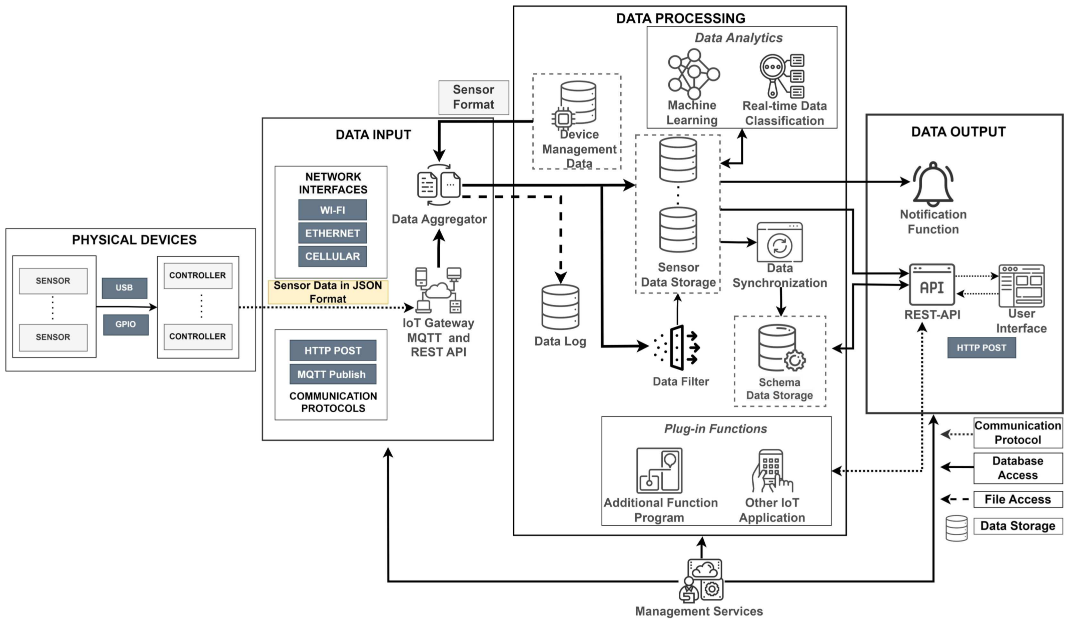


Figure 3.1: Design overview of *SEMAR* IoT application server platform.

3.2 Data Input

SEMAR needs to collect data from a number of different devices using various network connectivity and communication methods. Therefore, the following network interfaces for constructing physical network connections are implemented in the platform, where standard IoT communication protocols for data transmission, such as *HTTP* and *MQTT*, are included.

In the context of IoT, physical devices as a perception layer consist of a number of sensors connected to a controller. With the growth of IoT technology, controllers such as Arduino and Raspberry PI have provided diverse network connectivity to accept data from various sensors.

General Purpose Input Output (GPIO) is the programmable interface in the device controller to receive or send command signals from/to IoT sensor devices [29]. In IoT application systems, GPIO is the standard interface for connecting sensor devices with the controller.

Universal Serial Bus (USB) is the serial communication media to link devices with computers through USB ports. Currently, numerous sensor instruments and devices can transmit data using USB connections. The USB connection offers a high data transfer capacity.

In addition, external communication modules such as Wi-Fi for data communication can also be added using a USB connection.

Regarding the concept of IoT data transmissions, diverse hardware and software connectivity should be considered. Diverse network interfaces utilize hardware-based transmissions, such as Wi-Fi, Ethernet, Cellular, and LPWA(Low Power Wide Area), which enable machine-to-machine and device-to-server communication. *IEEE 802.11 wireless LAN (Wi-Fi)* is the most prevalent network interface in IoT systems. It connects devices with each other and to servers. Wi-Fi is useful for connecting a lot of devices regardless of their locations with computers, which improves IoT application developments. Ethernet offers secure and dependable *wired* connectivity. It is one of the most used network interfaces in IoT systems; however, the implementation can be difficult over long distances.

Although Wi-Fi and Ethernet offer excellent network performance, their security and coverage areas should be considered. The alternative network interfaces that can be utilized are *cellular* and *LPWA* networks. *Cellular* is the network interface allowing the mobility of devices with the existing widespread availability of *cells* to connect with the internet. Currently, 5G cellular connections offer solutions with wider bandwidths than Wi-Fi or Ethernet. *LPWA* technology introduces long-range, low power consumption, and higher-throughput communications [30]. It enables devices to communicate over long distances. Due to this coverage area and low power consumption, they are well-suited for IoT applications. *Sig-Fox*, *LoRa*, and *Narrowband IoT (NB-IoT)* are widely used *LPWA* technologies in IoT domain [31]. To establish communication, a transmitter and receiver devices are required.

As part of *Data Input*, the communication protocols are essential for handling data communication between IoT devices and servers. An IoT server should support publish-subscribe and push-and-pull messaging systems for sending and receiving data. Thus, our proposed system utilizes *MQTT* and *REST API* for the communication protocol services.

MQTT is one of the protocols that have been designed for data communications in IoT application systems. It can work with minimal memory and the processing power [32]. The *MQTT broker* works for receiving messages from clients, filtering the messages according to a topic, and distributing the messages to subscribers [33]. The *MQTT broker* is implemented in the IoT gateway function of the platform to provide data communication services in *SEMAR*. The IoT gateway function offers communication services to connect sensor devices to the server. By using this protocol, sensor devices can transmit messages containing sensor data in the JSON format with *MQTT* topics. By subscribing data at the same *MQTT* topic, the data aggregation program in the platform obtains each sensor data.

The *IoT gateway* function also implemented the *REST API* for receiving sensor data through the *HTTP POST* communication protocol. It can only receive data in the JSON format. The *REST API* provides URLs for sensor data transmission. The *management* function in the platform creates a unique URL for each device. The *HTTP POST* communication protocol is compatible with standard network interfaces. By using *REST API*, sensor devices can transmit data in the JSON format.

In an IoT system, the data lifecycle begins with the communication gateway receiving sensor data, continues with data aggregation and preprocessing, and concludes with data storage. For this purpose, *SEMAR* provides a *data aggregator* function. The *data aggregator* is the function for collecting data from various data sources, applying the value-added processing, and repackaging the information in a consumable format. Algorithm 1 illustrates the data processing procedure in this function. It forwards the result to the following data filter or stores it in the data storage through the database access.

Algorithm 1 Data aggregator.

```
Input  : Raw sensor data received through a communication protocol ($RSensor$)
         Device code ($Dcode$)
Output: Sensor data in a consumable format ($MSensor$)
begin
    Save $RSensor$ in a log file
    Convert $RSensor$ to JSON object
    Find the sensor format from the database using $Dcode$ as $Sformat$
    if $Sformat$ not empty then
        Initialize $MSensor \leftarrow$ empty JSON object
        for each item in $Sformat$ do
            if item in $RSensor$ then
                Set $MSensor[item] \leftarrow RSensor[item]$
            end
        end
        Set $MSensor["time"] \leftarrow currenttimestamp$
        return $MSensor$
    end
end
```

3.3 Data Processing

The *data processing* in the *SEMAR* server platform offers various functions. The large amount of data from *data input* will be processed to obtain meaningful information using some functions. The functions are implemented as independent modules to reduce system crashes at system failures. They can be extended to *microservices* [34, 35]. The concept of *microservices* is the method of developing a large-scale system with a set of small independent services. For their implementations, thread-based programs are adopted to improve their performances for real-time data processing. Each service will initiate a new thread to process the newly coming data.

3.3.1 Data Management (Storage and *Plug-in* Functions)

The data management system is the main function of the IoT platform. In the context of IoT, systems must provide data storage, transaction management, query processing, and data access for application systems. Thus, the IoT platform must offer services to process the data flow from input to output. Moreover, towards developing diverse IoT applications, devices involved in IoT should be able to generate different kinds of data types according to the application.

In order to provide various IoT application systems, *SEMAR* should be a useful platform for a variety of IoT application systems. Therefore, it needs to support massive amounts of data in various formats. Furthermore, it should be able to store all necessary data by providing data storage for each application. The *management data storage* is the database that stores the operating parameters in the *SEMAR* server platform including the implemented IoT application systems. The data includes information about connected devices, communications, and parameters for process functions running on the platform. On this platform, each device has its own unique sensor format. The management data storage database keeps the sensor format as a template to support the development of an IoT application system on this platform.

Meanwhile, the *sensor data storage* is the database that stores all the sensor data in the platform. In IoT application systems, sensor devices may offer various data and it may change it over time with unstructured formats. For this purpose, the platform uses the *Big Data* technology to store unstructured JSON objects, generating the unique data store for each device. This data storage only accepts data from registered devices. Therefore, I implemented an additional data storage system in the form of log files. *Log Files* is designed to keep the values of any defined or undefined data using the CSV format. Defined data represents sensor data that matches the format registered in the management data storage. Undefined data represents data whose format is not registered.

The *schema data storage* is a database designed to help users create their own data storage. It allows users to dynamically specify names, fields, and data types, making data storage more efficient and flexible. It supports multiple data types, including integer, float, date, time, date-time, and string. Figure 3.1 illustrates that this database is used to store data synchronization results. Through the *REST API*, other systems can access the sensor data storage. As the advantage of this database, it can be dynamically defined and modified by the user, supporting the integration of various complex IoT application systems.

The data management system plays a role in the sensor data storage process and provides access to additional data processing functions. Those services are not only for systems integrated into the IoT platform (*built-in*) but also for *plug-in* functions that may be deployed as an extension. This is important because an IoT application system may require unique data processing that is not implemented in the platform. Therefore, the platform is designed and implemented to facilitate *plug-in* functions to address these requirements. As a result, users can easily implement *plug-in* functions without modifying existing code. The *plug-in* functions can access the data in the platform through the *REST API*.

3.3.2 Data Filter and Synchronization

In this research, I additionally explore the data processing capabilities required by IoT applications that are not included in the standard data management services. For example, sensors of IoT devices may generate measurement errors and noise during the measuring process. It can impact the risk of data analysis problems. In addition, IoT applications such as indoor localization systems require real-time sensor data from several devices simultaneously. Therefore, our platform deploys the data filter and synchronization functionalities for processing sensor data.

The *data synchronization* function can synchronize the data from different devices by referring to the timestamp in the data store schema. The *timestamp* was given when the platform receives the data from the sensor device. Thus, the platform requests the data from each sensor's storage at a specified detection time.

Algorithm 2 illustrates the data processing procedure in this function.

Algorithm 2 Data Synchronization

```
Input  : Detection time ($Dtime$), List of sensor data will be synchronized ($LSensor$)
Output: List of synchronized data ($SyncData$)
begin
    Set $TimeStart \leftarrow currenttime$
    Set $TimeEnd \leftarrow TimeStart + Dtime$
    while $True$ do
        if $TimeEnd = currenttime$ then
            Set $DataSource, IdentifierList, SyncData \leftarrow$ empty vector
            for $each\ sensor \in LSensor$ do
                Set $DSensor \leftarrow$ captured $sensor$ data between $TimeStart$ and $TimeEnd$
                Set $GroupData$ as empty vector
                for $each\ row\ in\ DSensor$ do
                    if $row[Fi]\ not\ in\ IdentifierList$ then
                        Append $row[Fi]$ to $IdentifierList$
                    end
                    Append $row[Fv]$ to $GroupData[row[Fi]]$
                end
                for $each\ i \in GroupData$ do
                    Set $DataSource[sensor][i] \leftarrow$ processed $GroupData[i]$ use the selected function
                end
            end
            for $each\ ID \in IdentifierList$ do
                Set $SyncItem \leftarrow$ empty vector
                Append $ID$ to $SyncItem["identifier"]$
                for $each\ sensor \in LSensor$ do
                    if $DataSource[sensor][ID]\ is\ not\ empty$ then
                        Append $DataSource[sensor][ID]$ to $SyncItem$
                    end
                    Append $sensor[default]$ to $SyncItem$
                end
                Append $SyncItem$ to $SyncData$
            end
            Stores $SyncData$ to the schema data storage
            Set $TimeStart \leftarrow currenttime$
            Set $TimeEnd \leftarrow Tstart + Dtime$
        end
    end
end
```

For each sensor data, the field for the identifier (Fi) to group sample data in a specific value, the field for the value (Fv) to be synchronized, the default value ($default$), and the four functions to process the data are prepared. The following functions are implemented to process the data:

- *Average*: it returns the average value of the data collected during the detection time.
- *Current*: it returns the last value among the data collected during the detection time.
- *Max*: it returns the highest value among the data collected during the detection time.
- *Min*: it returns the lowest value among the data collected during the detection time.

The functions of filtering sensor data before being saved in a data storage are implemented. Digital filters are adopted to reduce noise and inaccuracies in data. It processes sensor data using various digital signal processing techniques in real-time. The techniques include low, high, and band-pass filters based on *Butterworth* [36, 37] and the *Kalman* filter [38, 39]. Each technique is associated with specific coefficients and parameters that define the behaviour and characteristics of the filter. To facilitate the filtering process, I prepared functions for each filtering technique in *SEMAR*. Users are able to define the parameters through the user interface of *SEMAR*. Once the user designs a filter function, *SEMAR* stores it as a configuration file in the storage. Then, this configuration file serves as a parameter to perform data filtering functions.

The following procedure is applied for filtering data:

- It receives sensor data in a JSON format.
- It selects the sensor field's value to be filtered.
- It processes the selected sensor data using the designed filter function.
- It adds the field value in the JSON object with the filter result.
- It stores the JSON object in the database.

3.3.3 Machine Learning and Real-time Classification

One of the exploitation scenarios for the massive quantity of IoT data is its predictive capability by utilizing machine learning approaches. Several researchers approved the effectiveness of machine learning implementation in IoT applications [40, 41]. Therefore, I implement machine learning and real-time classification functions in *SEMAR*.

The machine learning algorithms are implemented to help data classifications. The *Support Vector Machine (SVM)* [42, 43] and *Decision Tree* [44–46] are implemented in this platform as standard machine learning algorithms in IoT application systems.

Decision Tree employs tree decisions including event outcomes, resource costs, and utility costs. It can create a data model for predicting outcomes by learning simple decision rules according to the data features. The data model structure consists of internal nodes representing an attribute, branches representing a decision rule, and leaf nodes indicating an outcome. Here, *C4.5*, *CART (Classification and Regression Trees)*, and *Naive Bayes Tree* are selected and incorporated into the platform as the most well-known machine learning algorithms [44]. *CART* is the binary recursive partitioning method that can handle both numerical and category data [44–46]. It can determine the impurity degree of acceptable data and build a binary tree in which each internal node provides two classes for the accepted attribute. The tree is formed by iteratively picking the attribute with the lowest *Gini* index. The *Gini* index for each node is calculated by the following equation [44]:

$$Gini(t) = 1 - \sum_{i=1}^{n} P(i|t)^2 \tag{3.1}$$

Support Vector Machine (SVM) is utilized as the regression and classification technique [47]. This approach has been used for the big data classification [43]. The SVM computes linear decision boundary lines that can separate the data for the labeled groups. The SVM decision boundary line is calculated by the following equation:

$$f(x) = \sum_{\forall i} y_i \alpha_i K(x_i, x) \tag{3.2}$$

where y_i represents the class label, α_i represents the learned weight, $K()$ represents the kernel function, x_i denotes the support vector, and x denotes the labeled training sample data. The kernel function is given by a collection of mathematical operations used to process the input data and convert it into the required format. The *radial basis function (RBF)* kernel is one of the common kernel functions in SVM. The following equation illustrates the formula of the (RBF) kernel:

$$K(x_i, x_j) = \exp(-\frac{d(x_i, x_j)^2}{2l^2}) \tag{3.3}$$

where l represents the length scale of the kernel and $d(x_i, x_j)$ denotes the Euclidean distance between x_i and x_j.

The machine learning algorithms allow the user to use the data stored in the data storage as the sample data. This module can generate a data model for the real-time data classification module. The real-time data classification function is implemented to analyze a huge amount of data from various sensor devices by periodically running the following procedure:

1. It loads the data classification model made by the machine learning algorithm.
2. It receives sensor data from the database.
3. It classifies data into classes by running the data model.
4. It stores results in the database.

The classification model can be created by each user separately. Moreover, the user can start or stop the real-time data classification at the user interface.

3.4 Data Output

Several output components, such as the user interfaces, the *data export*, *REST API*, and the *notification* function, are considered to use the data in the platform. The user interface is provided at the web browser to allow users to see the sensor and synchronized data by tables, graphs, or maps. The platform allows users to access the sensor data using the time of data receipt. It receives the sensor data in the JSON format by accessing *REST API*. The column in the table is formed automatically based on the sensor format of each device. The platform can generate the graph for each registered format sensor. Visualization maps will display the data in digital maps based on the GPS data. The *data export* feature is designed and implemented to allow users to download data in Excel, JSON, text, or CSV format at the specified time. Users can use this feature by accessing the user interfaces.

REST API is employed as a back-end system to access the sensor data. The sensor data is retrieved from the database and converted to the JSON format. It will be sent to the user interface and *plug-in* functions using *HTTP POST* communications. The platform can exchange and integrate data with other IoT application systems via *REST API*.

The *notification* function allows the user to define the threshold for each sensor data as the trigger of the message notification. If the value is over the threshold, the platform will send a notification. The platform offers two different communication services. First, it publishes a message to a specific topic using the *MQTT* communication protocol. Thus, the IoT application system can subscribe to the topic to receive the messages. Second, it delivers email notifications through the mail server service installed on the server platform. The user can dynamically define email recipients.

3.5 Management Service

The *management* service is used to manage all functions in the *SEMAR* platform. It includes the managements of users, devices, communications, schema data, synchronization functions, analytics, data filters, and notification functions. The management of users allows us to add users, set permissions, and restrict access to the devices.

The *device management* service provides the functions to register the devices and the sensors of the IoT application system. It allows managing the sensor format for each device dynamically. The platform can process, save, and display the data registered in the sensor format. For convenience, the *SEMAR* platform provides a template to add the device with the same sensor format easily. The schema data management allows users to create the schema database, define the field format, and manage the data.

The *management* service provides the functions to add, update, and delete settings for data synchronizations, data analytics, data filtering, and notifications. It allows the user to run and terminate the function service in the data process. All the configuration settings are saved as JSON objects.

3.6 Implementation of *SEMAR* IoT Server Platform

In this section, I present the implementation of the *SEMAR* IoT server platform. Table 3.1 shows the summary of the implementation.

In this implementation, the following two types of communication protocol services are implemented for data input. *Mosquitto* [48] is installed for the *MQTT* broker. It allows the platform to receive messages through various *MQTT* versions and supports connections from Wi-Fi, Ethernet, and Cellular network interfaces. Then, *REST API* is implemented based on Python programming and *Tornado* web server [49]. It allows the platform to receive messages through *HTTP POST* and supports connections from Wi-Fi, Ethernet, and Cellular network interfaces.

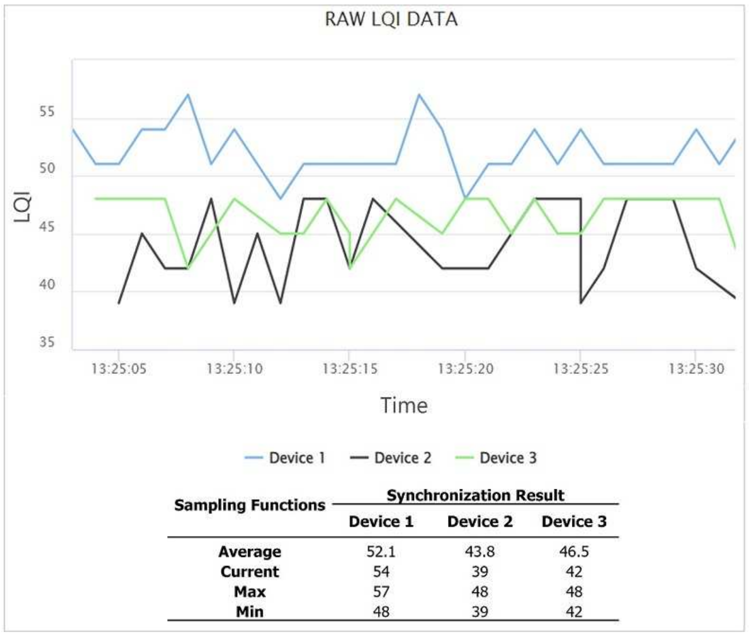
The data process is deployed and implemented in the platform. They are developed in Python using a variety of modules and dependencies. For IoT data management systems, I used two different database services implemented in the platform according to the design in Section 3.1. The Big Data repository *MongoDB* [50] is utilized for the data storage for managements, sensors, and schema. *MongoDB* saves data in the JSON format as the flexible approach. There is no need to define data structures, unlike *SQL*. In addition, the log file is implemented in the CSV format. It can be accessed using a file controller library in Python.

Two different data aggregators are implemented. The first one enables message receptions using the *MQTT* communication protocol. It allows different *MQTT* communication settings for each sensor device. The second one does it with *REST API*. Both data aggregators access the data storage via *PyMongo*.

Table 3.1: Technology specifications for implementation of *SEMAR* IoT server platform.

IoT Model	**Function**	**Component**	**Description**
Input	MQTT	MQTT Broker MQTT Supports	Mosquitto v2.0.10 MQTT v5.0, v3.1.1, and v3.1
	REST API	Libraries and Framework Communication Supports	Tornado Web Server, PyMongo, JSON HTTP-POST
	Network Interfaces	Network Interfaces Supports	Wi-Fi, Ethernet, Cellular
Process	Server	Operating System Memory	Ubuntu 18.04.5 LTS 6Gb
	Data Storage	Services	MongoDB v3.6.3
	Data Aggregator	Libraries and Framework Communication Supports	Tornado Web Server, PyMongo, JSON, Paho HTTP-POST and MQTT
	Data Filter	Libraries and Framework	PyMongo, JSON, Numpy, Scipy and KalmanFilter
	Data Synchronization	Libraries and Framework	PyMongo, JSON , Pandas, Statistics and Threading
	Machine Learning and Real-time Data Classification	Libraries and Framework	sklearn, Pandas, PyMongo, JSON, and Threading
Output	User Interfaces and Data Export	Programming Language Libraries and Framework Web services Development Pattern	PHP, CSS, HTML and Javascript CodeIgniter, Bootstrap, JQuery, HighChart JS, DataTables, OpenStreetMap Apache v2.4.29, PHP 7.2.24 MVC
	REST API	Libraries and Framework Communication Supports	Tornado Web Server, PyMongo, and JSON HTTP-POST
	Notification Functions	Libraries and Framework Notification supports Email Service	PyMongo, JSON, Paho, smtplib Email and MQTT Postfix
Management	Management Services	Libraries and Framework Communication Supports	Tornado Web Server, PyMongo and JSON HTTP-POST

In this thesis, the data filter and synchronization capabilities are utilized to process sensor data. *Scipy* and *KalmanFilter* Python libraries are used to apply the data filters. After filtering the data, *PyMongo* is used to save it in the data storage in JSON formats. The data synchronization used *PyMongo* for sensor data in the data storage. *Pandas* is used for grouping data sensors. *Threading* library is used to enhance the performance of the platform. This function runs periodically on the server based on the detection time. The user can stop and start this service at the administration page in the user interface. Figure 3.2 illustrates the user interface of the data synchronization function for the sensor data during 30 s.



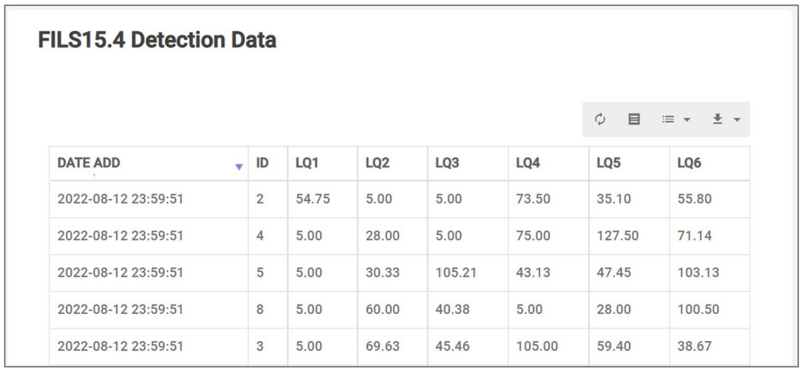
Sampling Functions	Synchronization Result		
	Device 1	Device 2	Device 3
Average	52.1	43.8	46.5
Current	54	39	42
Max	57	48	48
Min	48	39	42

Figure 3.2: Interface of data synchronization function.

According to the design systems in Section 3.1, the data analysis systems consist of learning process and real-time analysis service. I implemented both services in Python. *Scikit-learn* [51] is used to facilitate the learning process. The *Sklearn library* is utilized for real-time analysis to make the classification model during the learning process.

Data output includes the data visualization and the data sharing with other systems including the *plug-in* systems. The *CodeIgniter* PHP Framework is adopted to create user interfaces based on the *Model-View-Control (MVC)* design paradigm [52]. A user interface will offer data visualizations using *HighchartJS*, *DataTalbes*, and *OpenStreetMap*. Here, *Apache* and *PHP* are required. Figure 3.3 shows the table of sensor data. Figure 3.4 shows graphs of sensor data.

Utilizing the *DataTables* library in the user interfaces allows users to download sensor data in Excel, JSON, text, and CSV formats at the specified times. Figure 3.5 shows the data export interface. I built the *REST API* with Python and *Tornado*, enabling other application systems and *plug-in* functions to access sensor data in JSON formats.



FILS15.4 Detection Data

DATE ADD	ID	LQ1	LQ2	LQ3	LQ4	LQ5	LQ6
2022-08-12 23:59:51	2	54.75	5.00	5.00	73.50	35.10	55.80
2022-08-12 23:59:51	4	5.00	28.00	5.00	75.00	127.50	71.14
2022-08-12 23:59:51	5	5.00	30.33	105.21	43.13	47.45	103.13
2022-08-12 23:59:51	8	5.00	60.00	40.38	5.00	28.00	100.50
2022-08-12 23:59:51	3	5.00	69.63	45.46	105.00	59.40	38.67

Figure 3.3: Table of sensor data.

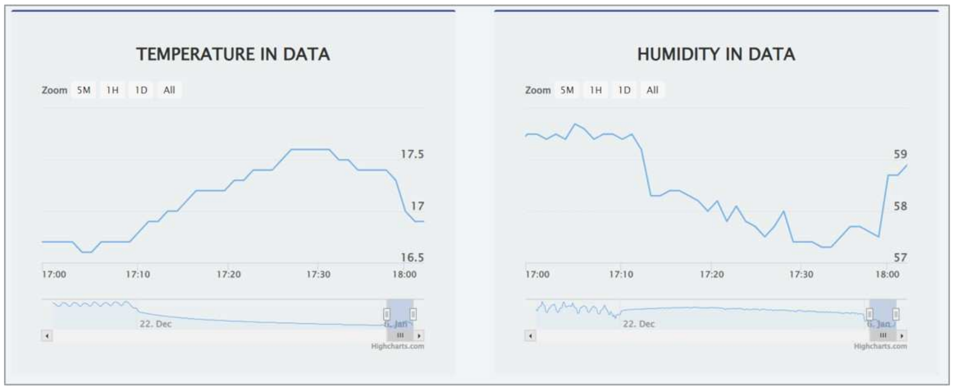
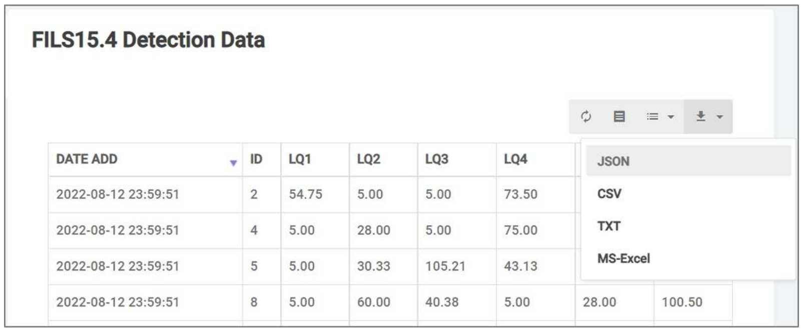


Figure 3.4: Graphs of sensor data.



FILS15.4 Detection Data

DATE ADD	ID	LQ1	LQ2	LQ3	LQ4		
2022-08-12 23:59:51	2	54.75	5.00	5.00	73.50		
2022-08-12 23:59:51	4	5.00	28.00	5.00	75.00		
2022-08-12 23:59:51	5	5.00	30.33	105.21	43.13		
2022-08-12 23:59:51	8	5.00	60.00	40.38	5.00	28.00	100.50

Figure 3.5: Data export interface.

Finally, the *management* service is built in Python and *Tornado* web server. It allows the platform to receive messages through *HTTP POST*, and to access data storage through *PyMongo*.

3.7 Integration of Air Quality Monitoring System

As the first IoT application system, the *air quality monitoring system* is integrated in the proposed platform. It can monitor the air quality in smart cities.

3.7.1 System Architecture

Figure 3.6 shows the system overview. This system uses a single-board computer (SBC) that is connected to the GPS sensor device and the air quality sensor device through Wi-Fi. The air quality sensor device covers the carbon monoxide sensor (MQ7), the particulate matter sensor (Shinyei PPD42), the sulfur dioxide sensor (MQ135), the ozone sensor (MQ131), and the nitrogen dioxide sensor (MiCS 2714. The sensor sends the voltage measurement data to the *Arduino UNO* via GPIO. *Arduino UNO* converts the data into the value of the pollutant concentration level and sends it to the SBC via the *MQTT protocol*. When the air sensor data are received, the SBC adds the current time and the location information (latitude and longitude) from the GPS sensor to the air sensor data, and sends it in the JSON format every five seconds through the *MQTT* connection.

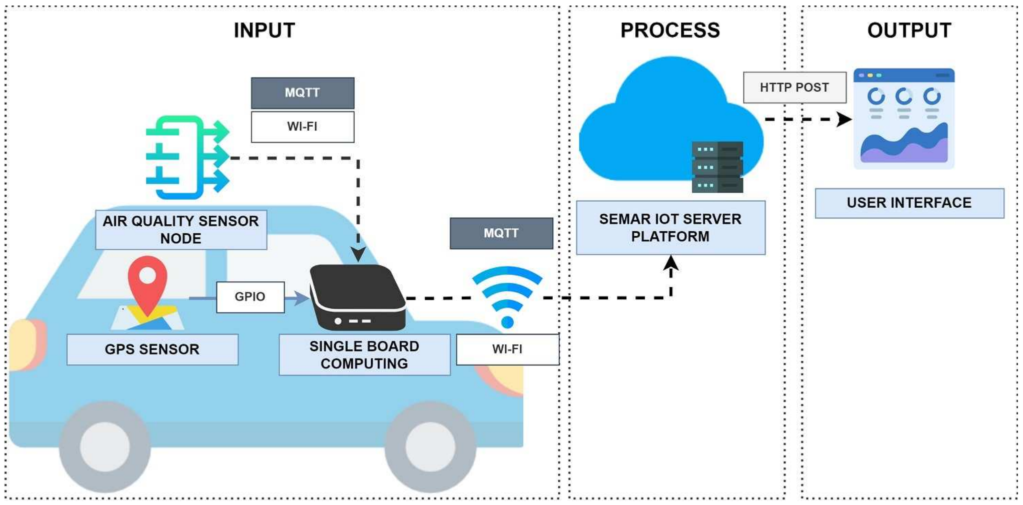


Figure 3.6: System overview of *air quality monitoring system*.

3.7.2 Implementation in Platform

Figure 3.7 shows the flow of the functions in the *SEMAR* server platform for integrating this IoT application system. Through the *MQTT* connection, the data aggregator receives the sensor data and stores it in the data storage. The real-time classification estimates the air quality index from the data between 0 and 4 that corresponds to the air quality categories of good, moderate, poor, very poor, and hazardous. The output data is shown at the user interface.

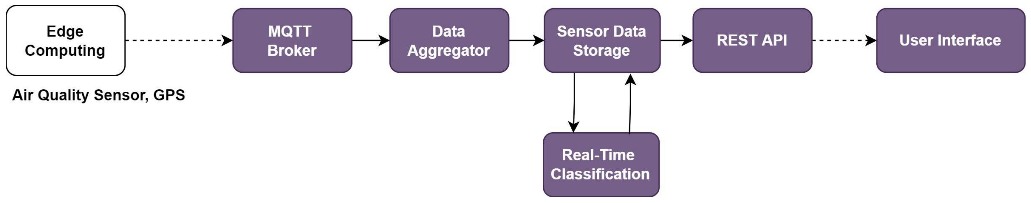


Figure 3.7: Function flow for *air quality monitoring system* in platform.

The integration was evaluated by running the system to monitor actual air quality conditions. The sensor device is mounted on the vehicle, and the single-board computer system is placed inside the vehicle during the experiment. The device system sends air quality and GPS data every five seconds. The evaluation results show that *SEMAR* has successfully received the sensor data, processed it, and classified the air quality index based on it. The results can be displayed on the user interface in real-time. Table 3.3 shows the evaluation results of the classification model used in this experiment. I compared two algorithms consisting of *Support Vector Machine (SVM)* and *Decision Tree (DT)*.

Table 3.3: Evaluation of air quality monitoring classification model.

Features	**Algorithm**	**Mislabel**	**Accuracy**	**MSE**
Air Quality	Support Vector Machine	605/10,053	0.94239	0.05761
	Decision Tree	43/10,053	0.99591	0.00409

Table 3.3 illustrates that the accuracy of the developed model is higher than 90%. Therefore, I can conclude that the real-time classification function to determine the air quality in *SEMAR* provides advantages over similar studies, including the study by Toma et al. in [53].

3.8 Integration of Water Quality Monitoring System

As the second IoT application system, the *water quality monitoring system* is integrated. It can monitor the water quality in rivers flowing in smart cities.

3.8.1 System Architecture

Figure 3.8 shows the overview of the system architecture. This system utilizes the sensor device equipped with water quality sensors for the hydrogen potential (pH), the oxidation reduction potential (ORP), the dissolved oxygen (DO), the electrical conductivity (EC), the temperature, total dissolved solids (TDS), the salinity (Sal), and the specific gravity (SG). The edge computing device *Raspberry Pi 3* collects the sensor data every five seconds and sends it to a server. The system was tested at various points in the river in Surabaya, Indonesia. The sensor node detects multiple parameters of water quality.

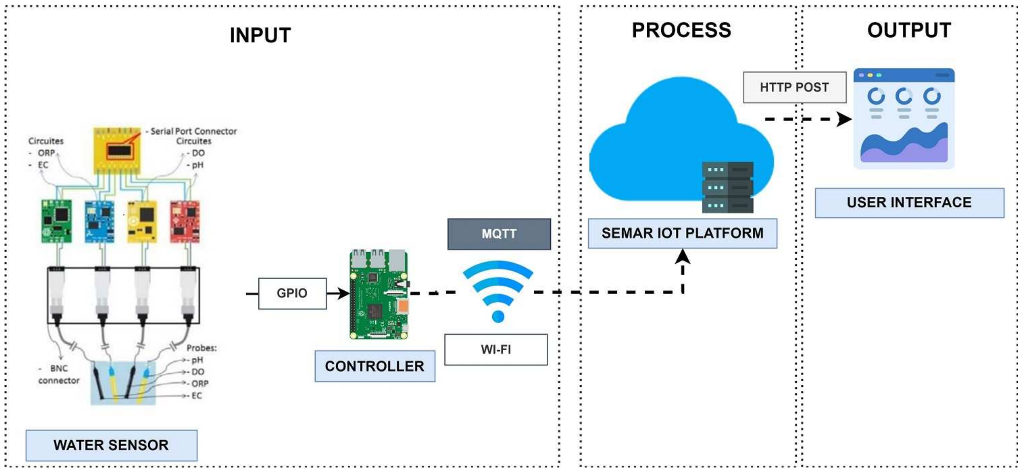


Figure 3.8: System overview of the *water monitoring system*.

3.8.2 Implementation in Platform

Figure 3.9 shows the flow of the functions in the platform for integrating this IoT application system. Through the *MQTT* connection, the data aggregator receives sensor data from the devices and stores it in the data storage. The real-time classification function estimates the water quality index from the collected data with a number between 0 and 3 corresponding to lightly polluted, heavy polluted, and polluted. The output data are shown in the user interface.

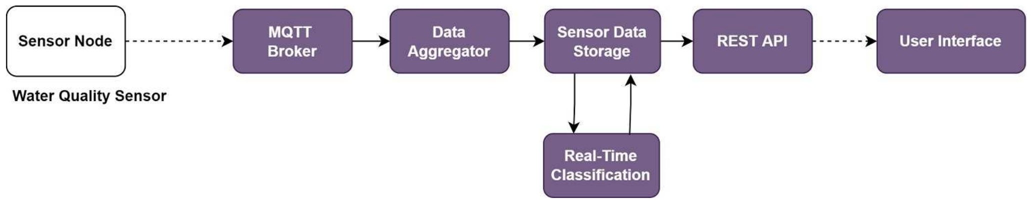


Figure 3.9: Function flow for *water quality monitoring system* in platform.

I evaluated the efficacy of the integration of *SEMAR* with the water quality monitoring system. The evaluation was conducted by operating the system in a real-world environment to monitor the water quality of a river. The device transmits the water sensor data to the *SEMAR* server every five seconds through *MQTT* communications. The experiment results indicate that the server received the sensor data, classified the water quality index based on the obtained data, and displayed it on the user interface in real-time. In addition, I compared the *SVM* and *DT* machine learning algorithms. Table 3.5 shows the evaluation results of the classification model utilized in the real-classification function.

Table 3.5: Evaluation of water quality monitoring classification model.

Features	**Algorithm**	**Mislabel**	**Accuracy**	**MSE**
Water Quality	Support Vector Machine	289/45,397	0.9936	0.0064
	Decision Tree	34/45,397	0.9993	0.0007

Table 3.5 shows that the accuracy of the classification model for the water quality is higher than 90%. Thus, the superiority of *SEMAR* on the integration with water quality measurement systems was confirmed with abilities to receive and classify data in real-time.

3.9 Integration of Road Condition Monitoring System

As the third IoT application system, the *road condition monitoring system* is integrated. It can monitor road surface conditions in smart cities.

3.9.1 System Architecture

Figure 3.10 shows the system architecture overview. This system is implemented as a mobile-based sensor network attached to the vehicle. This concept is called *Vehicle as a Mobile Sensor Network (VaaMSN)*. This system consists of an edge computing device, a portable wireless camera, and a sensor device. The camera records the road conditions in front of the vehicle and transmits the image frames through *Real-Time Streaming Protocol (RTSP)*. The sensor device collects GPS, accelerometer, and gyroscopes data, and transmits them to the edge computing device via *MQTT protocol*.

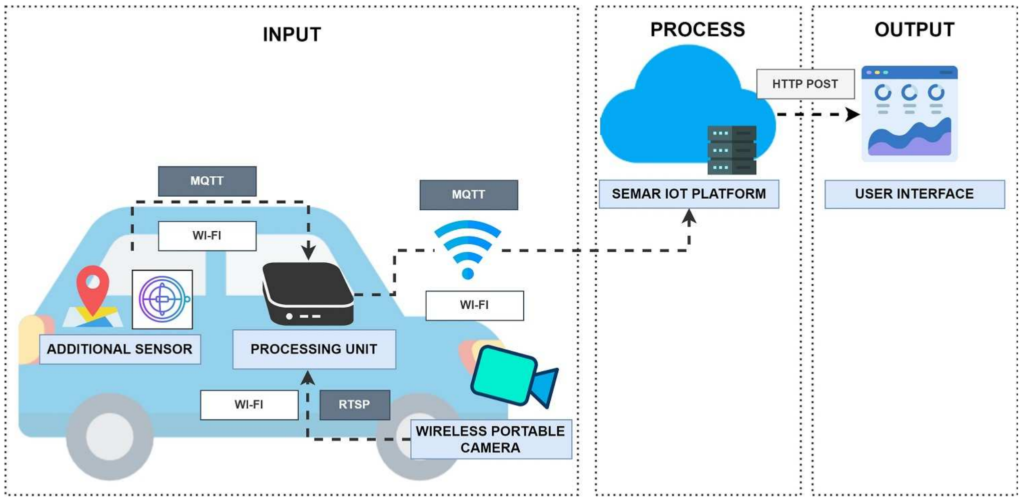


Figure 3.10: System overview of *road condition monitoring system*.

The edge computing device detects potholes from the camera images using the deep learning approach, *OpenCV* [54], and *Tensorflow* [55]. When detecting a pothole, image data are recorded in the directory file. Figure 3.11 shows the detected pothole example by the system. The edge computing will send the location, the accelerometer, the gyroscopes, and the pothole state to the server through the *MQTT* connection.

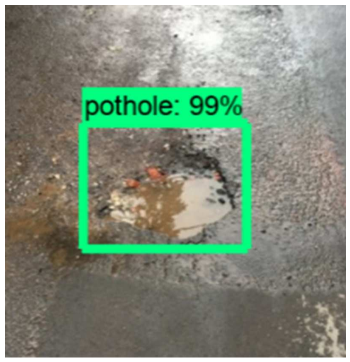


Figure 3.11: Detected pothole example.

3.9.2 Implementation in Platform

Figure 3.12 shows the flow of the functions in the platform for integrating this IoT application system. The data aggregator receives sensor data from the device through the *MQTT* connection and stores it in the data storage. The output data appear in the user interface.

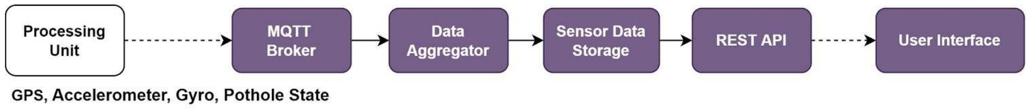


Figure 3.12: Function flow for *road condition detection system* in platform.

The integration was evaluated by running the system to monitor road surfaces in actual conditions. I place the sensor device in the vehicle according to the layout shown in the system overview. They send JSON data consisting of the GPS location, accelerometer, gyroscope, and pothole status to the server through *MQTT* communications when the system detects a pothole, as shown in Figure 3.11. The experiment results show that the system can receive data from the device, process it, and display it on the map of the user interface in real-time.

3.10 Integration of Air-conditioning Guidance System

As the fourth IoT application system, the *air-conditioning guidance system (AC-Guide)* is integrated. It can offer the guidance for the optimal use of air-conditioning (AC) in smart cities [56].

3.10.1 System Architecture

Figure 3.13 illustrates the system architecture overview. *AC-Guide* uses a web camera, a *DHT22* sensor, and *Raspberry Pi 3 model b+* as the sensor device. The Python program of the system periodically (1) collects the humidity and temperature of the room and the AC control panel photo, (2) collects the standard outdoor weather data by accessing to *OpenWeatherMap API* [57], (3) calculates the indoor *discomfort index (DI)* to determines whether the indoor state is *comfort* or *discomfort*, (4) calculates the *outdoor DI* to determines whether the outdoor state is *comfort* or *discomfort*, (5) detects the on/off state of the AC from the photo, (6) sends the message to *turn on* or *turn off* the AC considering the indoor DI, the outdoor DI, and the on/off state of AC, (7) saves the data in the log file, and (8) sends the data to the server using the *MQTT* connection.

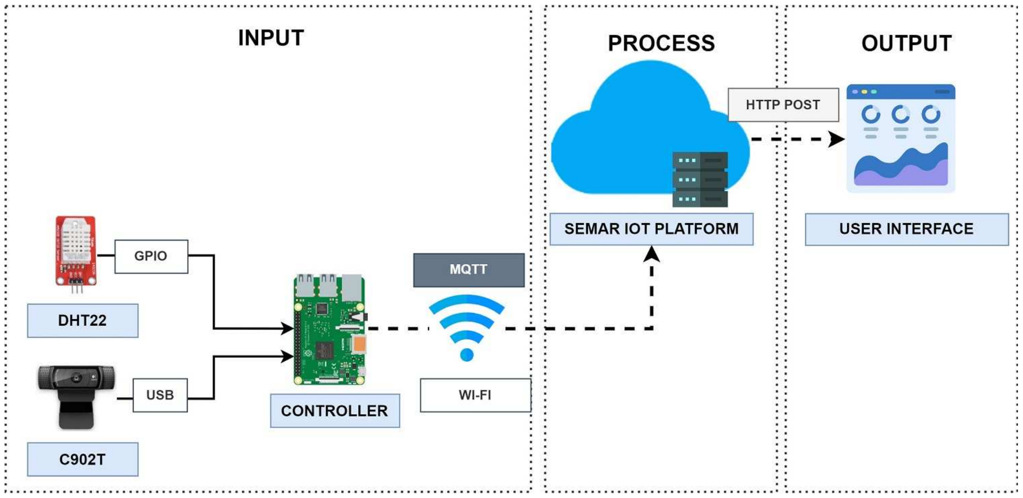


Figure 3.13: System overview of *AC-Guide*.

3.10.2 Implementation in Platform

Figure 3.14 shows the flow of the functions in the platform for integrating this IoT application system.

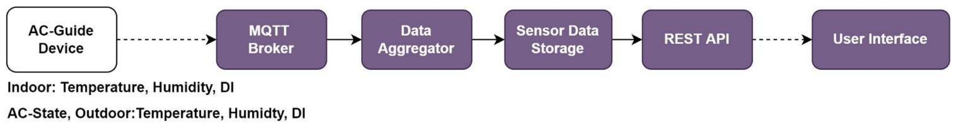


Figure 3.14: Function flow for *AC-Guide* in platform.

I evaluated the effectiveness of the integration of *SEMAR* with the air-conditioning guidance system. The experiment was carried out by running the system at the #2 Engineering Building in Okayama University. The device sends JSON data containing the indoor humidity, indoor temperature, indoor discomfort index (DI), outdoor humidity, outdoor temperature, outdoor *discomfort index (DI)*, and the state of AC using *MQTT* communications every one minute. The evaluation results show that *SEMAR* can receive sensor data and display sensor data in real-time on the user interface. Previously, these data were not accessible from other systems. By integrating *SEMAR*, they can access the data through *REST API*. In addition, *SEMAR* allows adding new sensors to the system without changing the codes. Therefore, the advantages of integrating the *SEMAR* system are confirmed.

3.11 Integration of Fingerprint-based Indoor Localization System

As the last IoT application system, the *fingerprint-based indoor localization system using IEEE802.15.4 protocol (FILS15.4)* is integrated. It detects the user locations in indoor environments according to the fingerprints of the target location. The process is divided into the *calibration phase* and the *detection phase* [58, 59].

3.11.1 System Architecture

Figure 3.15 illustrates the overview of FILS15.4 architecture. This system adopts transmitting and receiving devices by Mono Wireless which employs the *IEEE802.15.4* protocol at 2.4 GHz [60]. The transmitter *Twelite 2525* is small with 2.5×2.5 cm and can be powered with a coin battery for a long time. The receiver *Mono Stick* is connected to *Raspberry Pi* over a USB port. To improve the detection accuracy, the sufficient number of receivers should be located at proper locations in the target area.

Raspberry Pi receives data from a transmitter, determines the *link quality indication (LQI)* for each transmitter, sends the LQI with the ID to the *MQTT broker* using the *MQTT* protocol. The server receives them from the *MQTT broker*, synchronizes the data from all the receivers, calculates the average LQI with the same transmitter ID, and keeps the results in one record in the *SQLite* database. The previous implementation used a free public MQTT service.

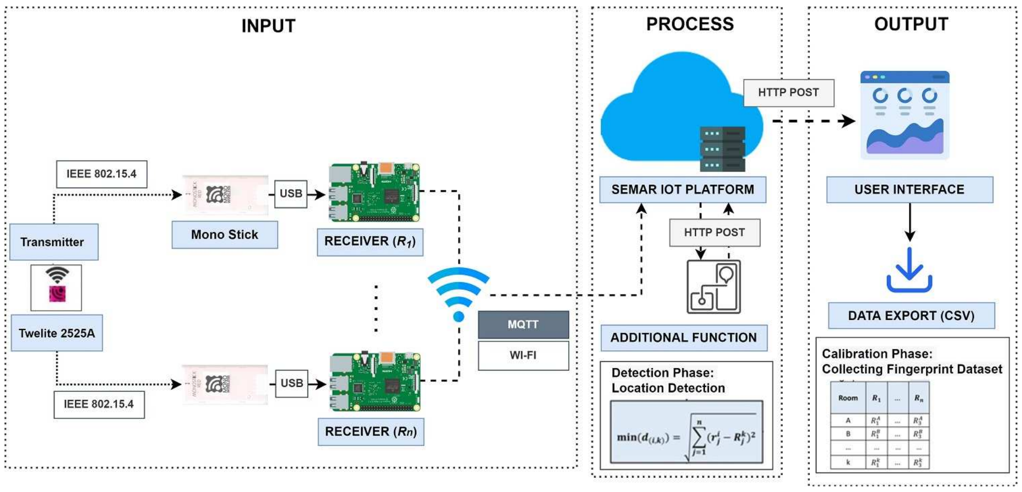


Figure 3.15: System overview of *FILS15.4*.

3.11.2 Calibration Phase

The *calibration phase* generates and stores the fingerprint dataset. Each fingerprint consists of *n* LQI values where *n* represents the number of receivers. It represents the typical LQI values when a transmitter is located at the corresponding location (room in *FILS15.4*).

3.11.3 Detection Phase

The *detection phase* detects the current room by calculating the Euclidean distance between the current LQI data and the fingerprint for each room and finding the fingerprint with the smallest distance.

3.11.4 Implementation in Platform

Figure 3.16 shows the flow of the functions in the platform for integrating this IoT application system. The data synchronization function synchronizes the measured LQI values among all the receivers using the transmitter's ID, and saves it in the schema data storage. The detection program is implemented as the *plug-in* function in the platform, and receives data through *REST API* services.

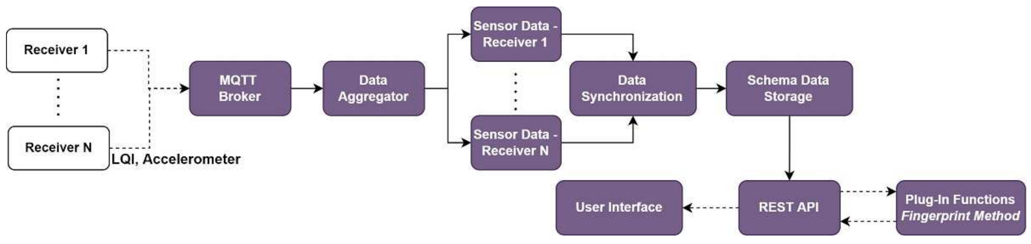


Figure 3.16: Function flow for *FILS15.4* in platform.

I evaluate the integration of *SEMAR* with the *fingerprint-based indoor localization system* by running the system at two floors in the #2 Engineering Building of Okayama University. This system used six receivers to measure LQI from each transmitter. The receiver sent the LQI data every 500 ms to the server through *MQTT* communications. The evaluation results show that *SEMAR* can receive, process, and visualize the data. I also evaluate the data synchronization of the LQI data at the multiple receivers from the same transmitter.

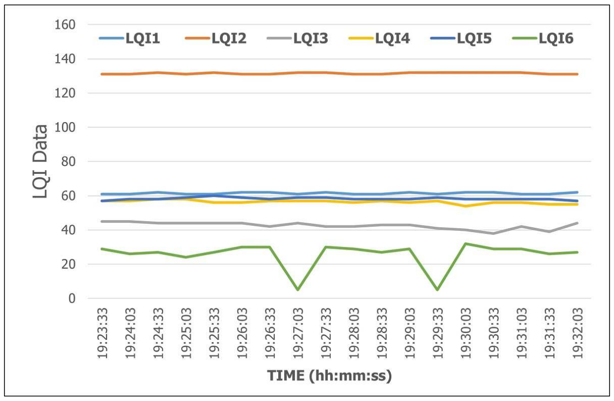


Figure 3.17: LQI data of *transmitter 1*.

Figure 3.17 shows the synchronized LQI data for *transmitter 1* during 30 s, where LQi for $i = 1, \ldots, 6$ indicates the LQI data at *receiver i*. They are saved in the schema data storage and can be accessed from other programs through *REST API*. This system can run without interruptions even if it processes empty LQI data or if error detection occurs. When the system detects an error, it sets the LQI data to the *default value*. According to the evaluation results, the effectiveness of integrating the *SEMAR* system is confirmed.

3.12 Evaluations of *SEMAR* IoT Application Platform

In this section, I evaluate the implementation of *SEMAR* application IoT server platform.

3.12.1 Performance Analysis

To evaluate the performance of *SEMAR* at the parameter level, first, I investigate the average response time for *MQTT* data communications when the number of IoT devices is increased from 1 to 125. In the experiments, a virtual IoT device is created in the system instead of a real device. Then, each virtual IoT device sends one message through a different topic every second. During this experiment, the CPU usage rate of the machine is also measured.

As the response time, the time difference at a virtual IoT device from the data transmission to the server to the message reception from the server is measured. For *HTTP POST*, it can easily be obtained. When the IoT device sends data to the server, the REST API service returns the response message; however, for *MQTT*, the program is modified to measure the response time where it will send the *MQTT* message to the device when it stores data in the storage.

Figures 3.18 and 3.19 show the average response time and the average CPU usage rate when the number of virtual IoT devices is increased from 1 to 125, respectively. The average response time is 315*ms* and the CPU usage rate is 74% for 125 devices. Thus, *SEMAR* our can handle hundreds of devices with acceptable delay and CPU rate.

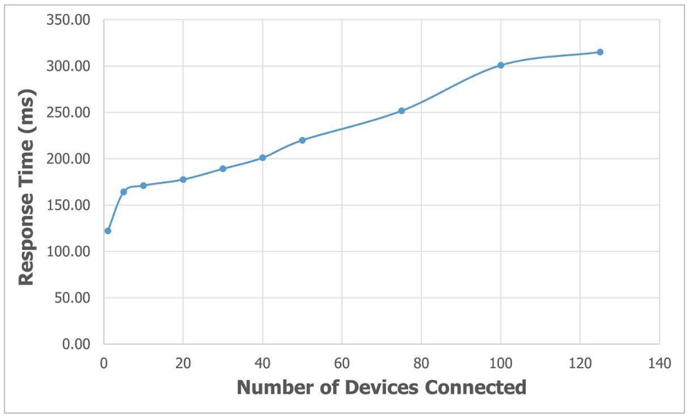


Figure 3.18: Average response time for *MQTT* communications with different numbers of devices.

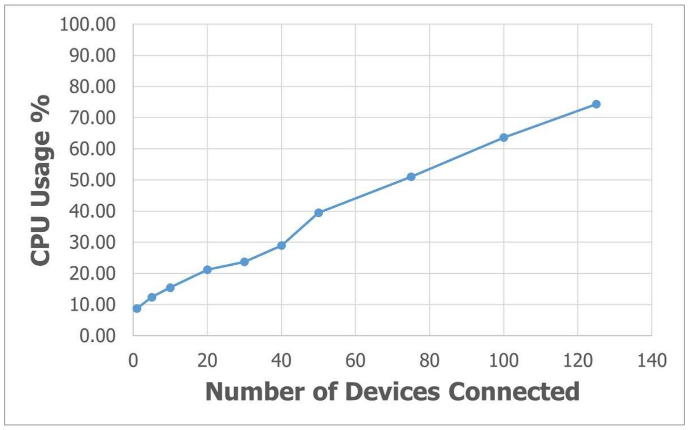


Figure 3.19: Average CPU usage rate with different numbers of devices.

3.12.2 State-of-the-Art Comparative Analysis

I compare the *SEMAR* IoT server platform with 14 recent research works that have the similar approach. In the comparison with the recent related works in the literature, I consider the following features to characterize each proposal:

- *IoT application*: represents the IoT application that is covered or implemented in each work.
- *Device management*: indicates the capability of the IoT platform to manage devices (Yes or No).
- *Communication protocol*: describes the communication protocol utilized in each work.
- *Data synchronization*: implies the capability to synchronize data across several devices (Yes or No).
- *Data filtering function*: indicates the implementation of digital filters to process data (Yes or No).
- *Decision-making assistance*: indicates the implementation of tools to evaluate data or generate alerts based on data obtained (Yes or No).
- *Flexibility*: shows the abilities to allow to join new devices, to handle different communication settings, to define data types, and to easily interact with external systems (Yes or No).

- *Interoperability*: represents the ability to be integrated with plural external systems through defined protocols (Yes or No).
- *Scalability*: demonstrates the capability of processing a number of data simultaneously (Yes or No).

Table 3.7 compares the fulfillment of the nine features among the 14 related works and the proposed *SEMAR*.

Table 3.7: State-of-the-art comparison between the existing related studies and the proposed solution.

Work Reference	**IoT Application**	**Device Management**	**Data Synchronization**	**Data Filter**	**Decision-making assistance**	**Flexibility**	**Interoperability**	**Scalability**	**Communication Protocol**
[61]	Indoor Air Quality	✓	✗	✗	✗	✓	✗	✓	HTTP
[62]	Smart Agriculture	✓	✗	✗	✓	✓	✓	✓	MQTT
[63]	Air Pollution	✓	✗	✗	✓	✗	✗	✓	HTTP
[64]	Water Management	✓	✗	✗	✗	✗	✗	✓	HTTP
[65]	Water Management	✓	✗	✗	✓	✓	✓	✓	MQTT
[53]	Air Pollution	✓	✗	✗	✗	✓	✓	✓	MQTT
[66]	Indoor Air Quality	✓	✗	✗	✓	✗	✓	✓	MQTT
[67]	Smart City	✓	✗	✗	✗	✗	✗	✓	HTTP & AMQP
[68]	Smart Industry	✓	✗	✗	✓	✓	✓	✓	MQTT
[69]	Smart Agriculture and Smart City	✓	✗	✗	✗	✓	✓	✓	MQTT
[70]	Smart Farming	✓	✗	✗	✓	✓	✓	✓	MQTT
[71]	Smart Building	✓	✗	✗	✓	✗	✓	✓	HTTP & Web Socket
[72]	Smart Irrigation	✓	✗	✗	✓	✗	✗	✓	MQTT
[73]	Smart Green and Smart City	✓	✗	✗	✗	✓	✓	✓	HTTP, MQTT, AMQP
SEMAR	Various IoT applications	✓	✓	✓	✓	✓	✓	✓	HTTP & MQTT

IoT Application

Although the works by Hernández-Rojas et al. in [62], Marcu et al. in [69], and Antunes et al. in [73] have potentials of use in various IoT applications, they have been studied in specific IoT applications. On the other hand, *SEMAR* has been integrated and implemented in several types of IoT applications.

IoT Device Management

All the related works provide functions to add or remove IoT devices. Some works support device management services. Some works include capabilities to define the sensor format for each IoT device dynamically. The work by Trilles et al. in [70] provides the easy-to-use user interface to manage IoT devices. On the other hand, *SEMAR* provides all of the functions on IoT devices.

Communication Protocol

HTTP and *MQTT* are the most adopted communication protocols in IoT application platforms. In addition, Del Esposte in [67] and Antunes in [73] introduce *AMQP* as another protocol utilizing TCP connections. Thus, it is suitable for server-client communications [74]. None of the related works reported functions to synchronize data from several devices and digital filters to process sensor data. Only *SEMAR* provides both the data synchronization capability and digital filters to process data.

Decision Making Assistance

For decision-making assistance, a lot of works have offered functions for perspective data analysis based on collected data. The works by Mandava et al. in [63], by Kamienski et al. in [65], by Chiesa et al. in [66], and by Boursianis et al. in [72] applied machine learning algorithms for real-time classifications, and show the results for user interfaces. The work by Hernández-Rojas et al. in [62] utilized message notifications according to a specific data threshold. The work by Trilles et al. in [70] and our *SEMAR* included both of them.

Interoperability and Flexibility

Several works provided interoperability. The works by Hernández-Rojas et al. in [62], by Trilles et al. in [70], and *SEMAR* allow outer programs to process data without changing the existing program in the systems.

Some works consider the flexibility as the IoT application platform. The works by Hernández-Rojas et al. in [62] and by Trilles et al. in [70] provide the capability to dynamically define the sensor format and the data type for each device, similar to *SEMAR*.

However, any work cannot be connected with other *MQTT* servers. Only *SEMAR* flexibly allows users to use other *MQTT* servers, which will allow IoT applications to be easily integrated with *SEMAR*.

3.13 Summary

This chapter presented the design and implementation of the *SEMAR (Smart Environmental Monitoring and Analytical in Real-Time)* IoT application server platform to facilitate the development of a cloud layer of IoT application systems. It offers integration functions in *Big Data* environments. This includes *built-in* functions for data aggregations, synchronizations, and classifications with *machine learning*, as well as *plug-in* functions that access the data through *REST API*. The platform was implemented and integrated with five IoT application systems. The results confirmed the effectiveness and efficiency of the proposal.

Chapter 4

Implementation of Edge Device Framework in SEMAR

This chapter presents the design and implementation of the *edge device framework*, which allows users to remotely optimize the utilization of edge devices by configuring them through the *SEMAR* IoT application server platform. It operates in the initial, service, and update phases. The implementation details within IoT applications are thoroughly evaluated.

4.1 System Overview

In this thesis, I propose the *edge device framework* as a collection of tools to facilitate the development of edge computing systems. Figure 4.1 provides an overview of the integrated system of the edge device framework in *SEMAR*. It functions in three phases. In the *initialization phase*, it offers web services that enable the automatic downloading of the configuration file to the device via *HTTP* communications. In the *service phase*, it transforms data from various sensors into the standard data format, processes the data, and periodically transmits them to the server. For data processing in this framework, I utilize digital signal processing techniques to implement filtering functions. In the *update phase*, it remotely updates the configuration through *MQTT* communications.

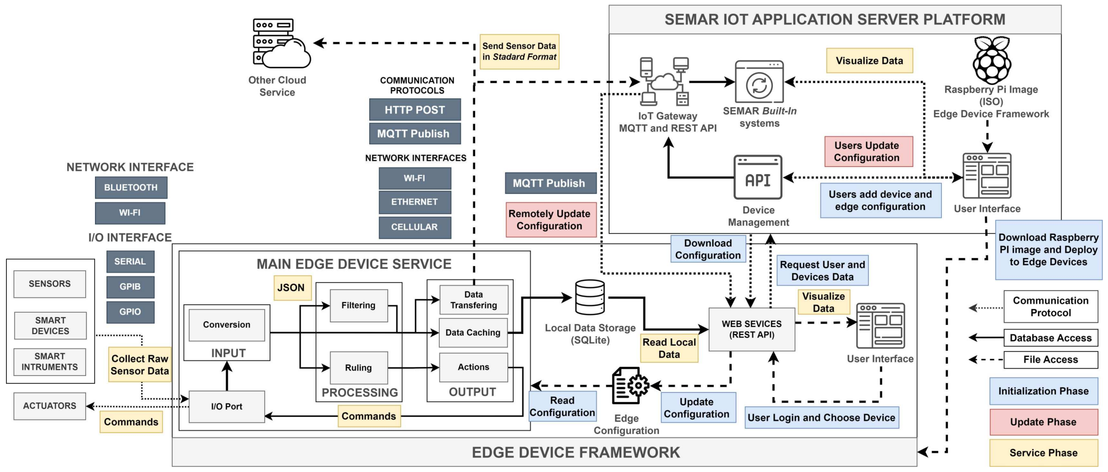


Figure 4.1: Design overview of the edge device framework.

4.2 Initialization Phase

In the *initialization* phase, the framework is installed on the edge device, and the initial connection is established between the edge device and the *SEMAR* platform. First, the user registers a new device and configures the edge device on the *SEMAR* platform via the user interface. Then, the user downloads the *Raspberry Pi* image from the *SEMAR* platform and deploys it to the edge devices. The user needs to ensure that the devices are connected to the Internet. Next, the user accesses the web services of the edge device framework through the user interface. The system verifies the user account by accessing the *REST API* services of the *SEMAR* platform. If the user account is authenticated, the system retrieves all the device data of the user from the *SEMAR* platform, generates the *edge ID* of the device, and grants the access to the web services.

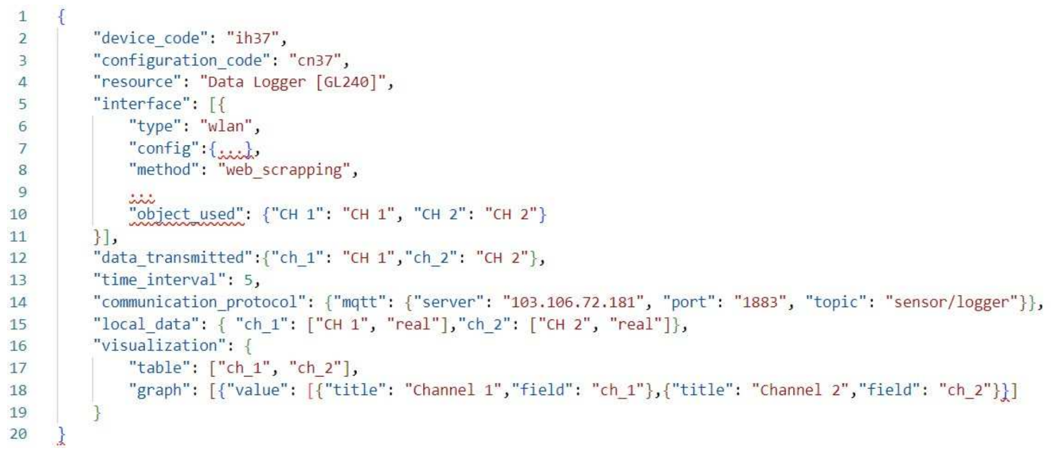
In the *initialization* phase, the user needs to choose the data to be applied to the edge device from the user interface. Then, the system downloads the edge configuration, saves it to the JSON file, and runs the *main service* program. Algorithm 3 illustrates the process flow of this program for both the *initialization* and *update* phases. Figure 4.2 shows the sample edge configuration file used in the framework. It includes the device, the device identity, and the configuration parameters such as the sensor interface, the data conversion method, the data model, transmitted data, the local data storage, the local visualization, and the filtering functions. The required libraries to run the system have been installed in the edge device framework.

Algorithm 3 Edge configuration service.

```
Input  : Edge ID (edgeID)
Output: Edge configuration file (EdgeConfig)
begin
    Set $EdgeConfig \leftarrow$ read $EdgeConfig$ from the "config.json"
    if $EdgeConfig$ not $NULL$ then
        Run Main Service program($EdgeConfig$)
        Connect to the MQTT broker in SEMAR
        Subscribe for the "edgeID" MQTT topic
        while true do
            if $Message \leftarrow$ receive data from server through MQTT communication then
                Set $EdgeConfig \leftarrow$ convert $Message$ to JSON format
                Save $EdgeConfig$ to the "config.json"
                Restart Main Service program($EdgeConfig$)
            end
        end
    end
end
```

4.3 Service Phase

In the *service phase*, which is the primary phase of the edge device framework, the framework collects and transmits sensor data to *SEMAR*. Figure 4.1 illustrates the lifecycle of the edge device framework for this purpose. Based on the general IoT application architecture illustrated in Figure 2.2, the functions of the main edge framework services are classified into *data input*, *data processing*, and *data output*.



```json
{
    "device_code": "ih37",
    "configuration_code": "cn37",
    "resource": "Data Logger [GL240]",
    "interface": [{
        "type": "wlan",
        "config":{...},
        "method": "web_scrapping",
        ...
        "object_used": {"CH 1": "CH 1", "CH 2": "CH 2"}
    }],
    "data_transmitted":{"ch_1": "CH 1","ch_2": "CH 2"},
    "time_interval": 5,
    "communication_protocol": {"mqtt": {"server": "103.106.72.181", "port": "1883", "topic": "sensor/logger"}},
    "local_data": { "ch_1": ["CH 1", "real"],"ch_2": ["CH 2", "real"]},
    "visualization": {
        "table": ["ch_1", "ch_2"],
        "graph": [{"value": [{"title": "Channel 1","field": "ch_1"},{"title": "Channel 2","field": "ch_2"}]]
    }
}
```

Figure 4.2: Sample edge configuration file in JSON format.

Algorithm 4 describes the program flow. To collect the raw sensor data, the edge device must be connected to the sensor or device. The service program then reads the edge configuration file, which was downloaded by the edge configuration services.

Algorithm 4 Service phase.

```
Input   : Edge configuration($EdgeConfig$)
begin
    Set $TimeInterval$, $CommService$, $Interface$, $TransmitData$, $FilterModel$, $RuleModels$,
      $LocalData$, $ActionModels$ ← read the configuration of time interval, communication service,
      resource interface, transmitted data from $EdgeConfig$
    Set $SensorResource$ ← connect to the network interface of sensor device($Interface$)
    while $true$ do
        Set $RawSensor$ ← read raw data of sensor from $SensorResource$
        Set $ConvertData$ ← convert raw data of sensor to the standard
          format($RawSensor$, $Interface$)
        if $FilterModel$ not empty then
            Set $ConvertData$ ← procces sensor data using digital filter($ConvertData$,$FilterModel$)
        end
        if $RuleModels$ not empty then
            Set $RullingResults$ ← applying rule models($ConvertData$,$RuleModels$)
        end
        Save sensor data to the local storage($ConvertData$,$LocalData$)
        Set $Data$ ← select transmitted sensor data($ConvertData$, $TransmitData$)
        Send transmitted data to the server through communication service ($Data$, $CommService$)
        if $RullingResults$ not empty then
            Send commands to control actuators($RullingResults$,$ActionModels$)
        end
        sleep($TimeInterval$)
    end
end
```

As illustrated in Algorithm 4, the program can process the raw sensor data by converting them to the standard data format, reducing inaccuracies in the data using the *filtering* function, generating the decisions based on predefined rule models using the *ruling* function, saving it to local data storage, and sending it to the server in JSON format using a defined communication protocol.

The communication protocol can be either *MQTT* or *HTTP POST*. The *SEMAR* platform receives, processes, and analyzes the sensor data using built-in systems on the server and displays the sensor data as output in the user interface. Additionally, the system can send notifications/alerts to the user and trigger actuators based on the rule model results. The program runs periodically at specific intervals and only transmits the sensor item values defined in the configuration file. Therefore, the framework enables the user to manage edge devices and optimize their performance by defining edge configuration files.

One difficulty in inputting data into the edge device framework involves the connectivity of the sensor interface. The aim of the edge device framework is to create a versatile edge computing device that can automatically gather and transmit sensor data to the server. Therefore, it is essential to establish connectivity services and data models that can support multiple sensors. Currently, the system can capture and transform sensor data through the GPIO, USB serial, and wireless interfaces. I have created multiple functions with which to collect data from the GPIO interfaces. To use the system, the user must first specify the GPIO ports and modes in the configuration file. Then, the system periodically reads the port value, converts it into a JSON object based on the configuration file, and returns the results to the data processing components.

To use the USB serial interface, the user needs to specify the serial port, the timeout time, and the baud rate that determines the data transmission speed. The user also needs to define the delimiter that the system will use to extract the relevant information when it receives a line of serial communication data. Algorithm 5 shows the data conversion process for serial communications.

Algorithm 5 Data conversion procedure for serial communication.

```
Input  : Raw sensor data ($RawSensor$), Edge configuration($EdgeConfig$)
Output: Converted sensor data ($ConvertData$)
begin
    Set $Delimeter, ObjectUsed$ ← read configuration of delimiter and object used, from $EdgeConfig$
    Initialize $ConvertedData, Result$ ← empty JSON object
    Set $DataList$ ← SPLIT($RawSensor$,$Delimeter$[0])
    for each item in DataList do
        Set $Buffer$ ← SPLIT($item$,$Delimeter$[1])
        Set $Result[Buffer[0]]$ ← $Buffer[1]$
    end
    for each sensor in ObjectUsed do
        if sensor in Result then
            Set $ConvertData[sensor]$ ← $Result[sensor]$
        end
    end
    return $ConvertData$
end
```

To use the wireless interface, the user needs to provide the URL of the web service to receive the HTML data through *HTTP GET* communications. The web scraping technique is used to extract the necessary information from the HTML data and to transform it into an array format. The user needs to define the index array that includes the channel name and sensor value. The data conversion process for the wireless interface data is shown in Algorithm 6, which illustrates the data conversion procedure for the wireless interface data.

Algorithm 6 Data conversion procedure for wireless interface.

```
Input  : Raw sensor data in HTML format ($RawSensor$), Edge configuration($EdgeConfig$)
Output: Converted sensor data ($ConvertData$)
begin
    Set $ChannelIndex, ValueIndex, MaxSequence, ObjectUsed$ ← read configuration from $EdgeConfig$
    Initialize $ConvertedData, Result$ ← empty JSON object
    Set $DataList$ ← WEBSCRAPING($RawSensor$)
    for $i \leftarrow 0$ to $length(DataList)$ do
        if $i$ % $MaxSequence == ChannelIndex$ then
            Set $ChannelName \leftarrow DataList[i]$
        end
        if $i$ % $MaxSequence == ValueIndex$ then
            Set $SensorValue \leftarrow DataList[i]$
        end
        if $i$ % $MaxSequence ==$ ( $Maxsequnce$ - $1$ ) then
            Set $Result[ChannelName] \leftarrow SensorValue$
        end
    end
    for each sensor in $ObjectUsed$ do
        if sensor in $Result$ then
            Set $ConvertData[sensor] \leftarrow Result[sensor]$
        end
    end
    return $ConvertData$
end
```

The data transfer system has been implemented to enable the transmissions of sensor data to not only the *SEMAR* platform but also to any other IoT gateway service the user prefers for the cross-vendor capability in edge computing. It currently supports *HTTP POST* and *MQTT* communications using the standard JSON format. The data transfer function uses the *"time_interval"* configuration to regulate the data transfer frequency, the *"data_transmitted"* configuration to determine the output data to be transferred, and the *"communication_protocol"* configuration to describe the destination and communication service. While developing edge devices, the communication network is the critical factor for avoiding the unsuccessful data transfer. The data caching function is implemented by using SQLite and Python to store sensor data locally, with the *"local_data"* configuration specifying which data are saved in the local data storage.

The current implementation allows the user to visualize data in the forms of tables and graphs. It is accomplished using the *"visualization"* setting, which retrieves sensor data from an *SQLite* database. The user can access these data through the web interface or the *REST API* service. To make IoT application system developments more flexible, I utilized the *REST API* service at the edge layer to integrate edge device frameworks with other systems.

4.4 Update Phase

In the *update phase*, the user has the ability to remotely modify the edge configuration file on the edge device using the *SEMAR* user interface. This process involves modifying the edge configuration and utilizing the deploy button to initiate the remote update function. The *device management* service transmits the updated edge configuration in the JSON format to the relevant edge device using *MQTT* communications with the edge ID as the topic. The edge configuration service connects to the *MQTT* broker within *SEMAR* and subscribes to the same topic with the edge ID. After receiving the new edge configuration through *MQTT* communications, the service saves it in the designated folder and triggers the function to restart the service program. As a result, the user can easily add new sensor devices or modify device configurations by making adjustments through the user interface. Figure 4.3 illustrates the flow process of the *update phase*.

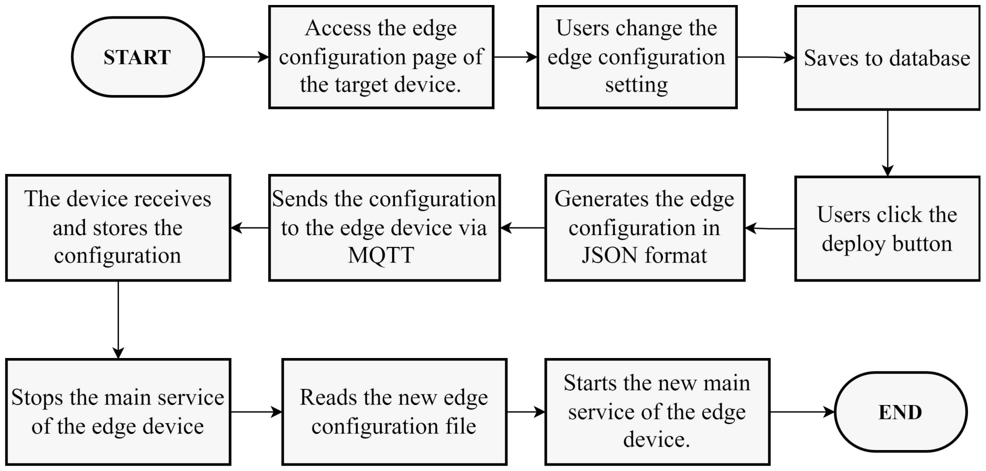

START
Access the edge configuration page of the target device.
Users change the edge configuration setting
Saves to database
Users click the deploy button
Generates the edge configuration in JSON format
Sends the configuration to the edge device via MQTT
The device receives and stores the configuration
Stops the main service of the edge device
Reads the new edge configuration file
Starts the new main service of the edge device.
END

Figure 4.3: Flow diagram of *update phase*.

4.5 Filtering Functions

This section presents the *filtering* functions based on digital signal processing techniques in the *edge device framework*.

4.5.1 System Overview

Figure 4.4 illustrates the system overview of the *filtering* function in the *edge device framework*. It includes *digital filter*, *cascading filter*, and *aggregating* functions. The *digital filter* implements digital signal processing techniques to filter the sensor data received from the input functions. The *cascading filter* creates *digital filter* functions in a sequential manner. Each *digital filter* and *cascading filter* function is responsible for processing a single field of the sensor data and runs in parallel. The *aggregating* function collects the filtered data, combines them with the sensor data, and sends them to the output functions.

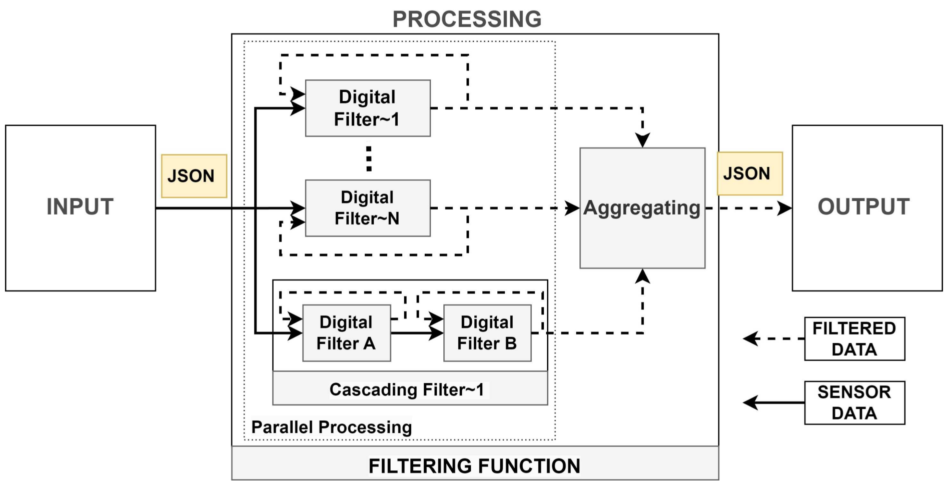
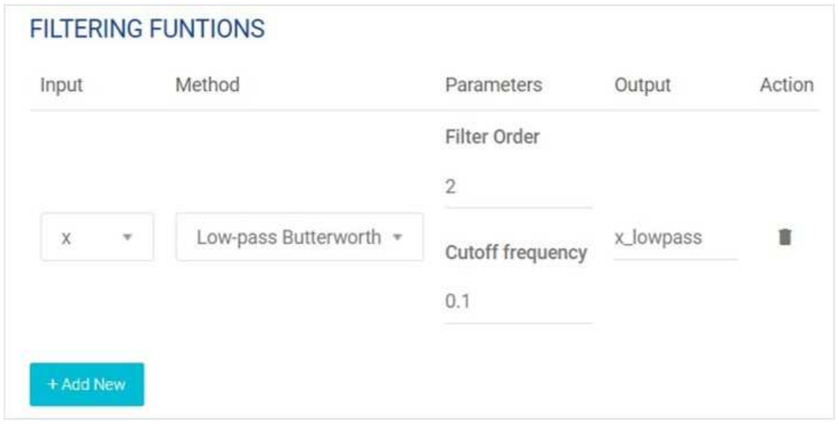


Figure 4.4: System overview of *filtering function*

4.5.2 Digital Filter

The *digital filter* processes input data using various digital signal processing techniques in real-time. The techniques include low, high, and band-pass filters based on *Butterworth* and *Chebyshev-I*, as well as *Kalman* and *Savitzky-Golay* filters. Each technique is associated with specific coefficients and parameters that define the behavior and characteristics of the filter. To facilitate the filtering process, I prepared functions for each filtering technique. These functions are implemented in the *edge device framework* and can be called during the filtering process performed by the digital filter.

To enhance the adaptability of the system, users are allowed to specify the filtering techniques, set the parameters, choose the sensor data fields to be processed, and assign output variable names for the filtered values. Through the *SEMAR* user interface, users can modify the configuration file to customize the digital filter to their specific requirements. This flexibility allows users to achieve the desired filtering results and adapt the framework to their unique application scenarios. Figure4.5 illustrates the user interface for configuring the filtering functions in the *SEMAR* platform.


FILTERING FUNTIONS

Input	Method	Parameters	Output	Action
x	Low-pass Butterworth	Filter Order: 2; Cutoff frequency: 0.1	x_lowpass	

Figure 4.5: User interface of *SEMAR* for configuration of filtering functions.

The following digital signal processing techniques are implemented in the *filtering* function.

4.5.2.1 Butterworth and Chebyshev-I Filters

The *Butterworth* and *Chebyshev-I* filters are digital signal processing techniques often used for data filtering. In the IoT context, *Butterworth* filters are used to smooth a time series data of sensors by removing noise at the signal level [36, 37]. This is achieved by keeping the signal spectrum based on a specified cutoff frequency. The response of *Butterworth* filters in the frequency domain is given by Equation (4.1).

$$|H(jw)| = \frac{1}{\sqrt{1 + (\frac{w}{w_c})^{2N}}} \tag{4.1}$$

where $H(jw)$ indicates the response magnitude, j indicates the imaginary unit, w and w_c indicate the input and cutoff frequencies, and N indicates the filter order [75].

Compared to *Butterworth* filters, *Chebyshev Type I* filters consider the ripple factor in the passband frequencies while filtering the input signal [76]. This strategy reduces the error between the desired and actual frequency responses. The response of *Chebyshev Type I* filters in the frequency domain is given by Equation (4.2).

$$|H(jw)|^2 = \frac{1}{\sqrt{1 + \varepsilon^2 T_N^2(\frac{w}{w_c})}} \tag{4.2}$$

where ε indicates the ripple factor, T_N indicates a *Chebyshev* polynomial of the N filter order [77].

In this study, I employed a customized approach to implement these techniques for real-time data filtering. This approach involves the filter design, the filter coefficient calculation, and the sensor data filtering using the difference equation.

The filter design was performed by defining the desired filter type, order, and cutoff frequency. The filter type includes low, high, and band-pass filters. *Butterworth* filters only consider these parameters, while *Chebyshev-I* filters need to define ripple factor parameters due to their ability to control variations in the passband frequencies [76]. Therefore, the framework was designed to allow users to define these parameters through the edge configuration file.

The characteristics of the designed filter used for data filtering are determined by the filter coefficients. These characteristics include feedforward coefficients (b) and feedback coefficients (a). For this purpose, the *SciPy* library [78] provided by *Python* is used. It generates an array of b and a according to the designed filters. The lengths of the b and a arrays depend on the filter order (N), and both are similar.

I developed digital filter functions by utilizing the difference equation in Equation (4.3).

$$Y_t = (\sum_{i=0}^{K} b_{t-i}X_{t-i}) - (\sum_{i=1}^{K} a_{t-i}Y_{t-i}) \tag{4.3}$$

where Y_t and X_{t-0} represent the filtered value and the sensor value at the current time t, K represents the lengths of the b and a arrays, Y_{t-i} and X_{t-i} represent the list of the previous filtered values and sensor values. The equation (4.3) suggests that the previous filtered and sensor values are combined with the corresponding coefficients to create filter operations. In order to perform this computation, I implement the *FIFO (First in First Out)* method to temporarily store the previous values during the measurement.

4.5.2.2 Kalman Filter

The *Kalman* filter is a technique that can handle noise by estimating the state of a dynamic system. According to the purpose of this framework, I prepared a *Kalman* filter function for filtering one-dimensional data [38,46]. The function is composed of the *prediction* phase and the *update* phase. The *prediction* phase brings the estimated value of the current state according to the results of the previous state through the following equation.

$$Y_t^p = AY_{t-1} \tag{4.4}$$

$$P_t^p = AP_{t-1}A^T + Q \tag{4.5}$$

where Y_p^t and P_p^t are the estimates of the filtered and the error covariance data in the current state t. Y_{t-1} and P_{t-1} are the previous data of the filtered and the error covariance. A is the state transition matrix. Q is the process noise covariance.

The *update* phase combines the prediction result of $Y_t^p p$ with the current sensor value X_t to obtain a more accurate filtered value Y_t through the following equation.

$$K = \frac{P_t^p H^T}{HP_t^p H^T + R} \tag{4.6}$$

$$Y_t = Y_t^p + K(X_t - HY_t^p) \tag{4.7}$$

$$P_t = (1 - KH)P_t^p \tag{4.8}$$

where K is the *Kalman* gain, H is the measurement matrix, and R is the measurement noise covariance. According to equations (4.4) to (4.8), I define configuration parameters for the *Kalman* filter, which include the state transition matrix A, the process noise covariance Q, the measurement matrix H, and the measurement noise covariance R.

4.5.2.3 Savitzky-Golay Filter

The *Savitzky-Golay* filter provides the ability to remove noise from data using less computing resources [79]. It divides the data into small sections called moving windows. It involves curve-fitting polynomials to capture and estimate the data trend in each window. It then produces a smoothed representation that eliminates the noise. This process is performed by the following equation.

$$Y_t = \sum_{i=-m}^{m} c_i X_{t+i} \tag{4.9}$$

where Y_t is the filtered value at time t, X is the input data matrix in the chosen window, and c is the coefficients matrix of the curve fitting polynomial [80]. Since the window size is denoted by $2m+1$, m represents half of the window size (excluding the X_t value). A least-squares approach is used to generate the coefficients matrix c by considering the polynomial order and window size [81].

In general, the polynomial degree and window size parameters affect the performance of the filter. The polynomial degree determines the complexity of the fitting curve. The window size specifies the number of data points considered in each window. Therefore, I created a function for the *Savitzky-Golay* filter that allows these parameters to be defined by the user in the edge configuration file.

4.5.3 Cascading Filter

A *cascading filter* is constructed by connecting multiple individual *digital filter* functions in a sequential manner. These functions collaborate in a chain-like structure, where the output of one filter becomes the input of the next filter in the sequence. This arrangement enables the combination of multiple filters to perform more advanced and precise filtering operations. The system allows users to define the sequence of *digital filter* functions in the *cascading filter* and configures their settings to improve the filtering performance.

To facilitate this customization, I have improved the configuration file of the framework. Users are able to define the sequence of the *digital filter* functions used, the configuration of each *digital filter* function, the field of sensor data to be processed, and the output variable names for storing the filtered values.

4.5.4 Aggregating

Both the *digital filter* and the *cascading filter* functions can run in parallel. To collect the filtered data from them, an *aggregating* function is implemented. This function appends or merges the filtered data with the original input sensor data and creates a JSON data structure containing the filtered values. After the aggregation is completed, it sends the JSON data to the output functions of the edge device framework. The variable name used to store each of the filtered data is defined by the user in the configuration file.

4.6 Application for Fingerprint-Based Indoor Localization System

As a first application, I implemented the edge device framework to support the integration of the *FILS15.4* system into the *SEMAR* IoT application platform.

4.6.1 System Architecture

Previously, I integrated the *FILS15.4* system into the *SEMAR* server. In this implementation, I intend to apply the edge device framework to build receiver devices for the *FILS15.4* system. Figure 4.6 illustrates the overview of the implementation of the edge device framework into *FILS15.4* system.

As described in the subsection 3.9, the *FILS15.4* system utilizes the transmitter and receiver devices produced by *Mono Wireless* that operate on the *IEEE802.15.4* standard at 2.4 GHz [60]. The transmitter *Twelite 2525* is powered for a long time by a coin battery and sends data to the receivers. The receiver *Mono Stick* is connected to the *Raspberry Pi* through a USB connection. It receives data and determines the LQI for each transmitter. It then sends the data consisting of the LQI value, the transmitter ID, and the accelerometer data to the server through the *MQTT* communication protocol.

4.6.2 Evaluation of Implementation

The implemented edge device framework for *FILS15.4* was deployed on two floors in the #2 Engineering Building at Okayama University for evaluations. The evaluations intended to verify the

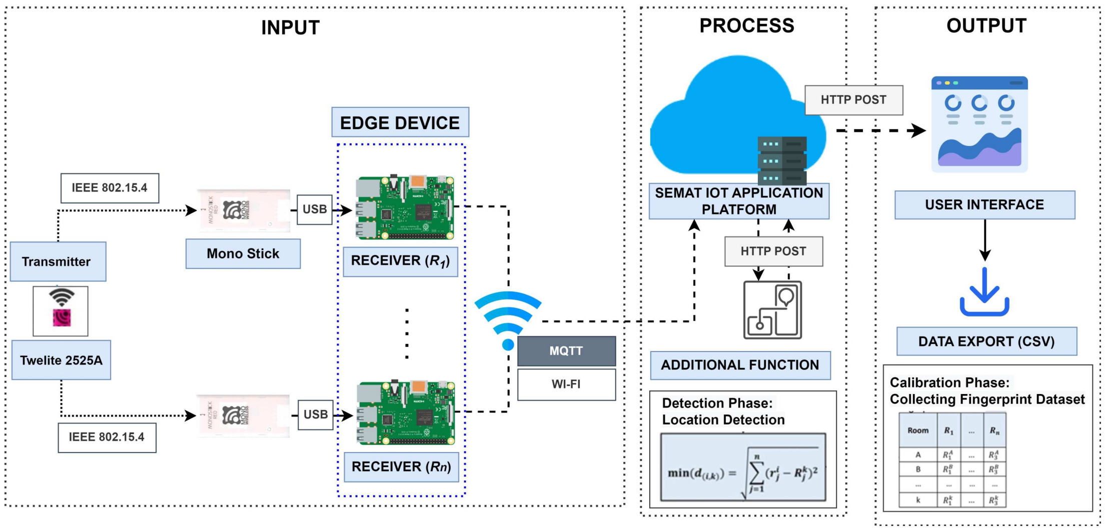


Figure 4.6: System overview of *FILS15.4*.

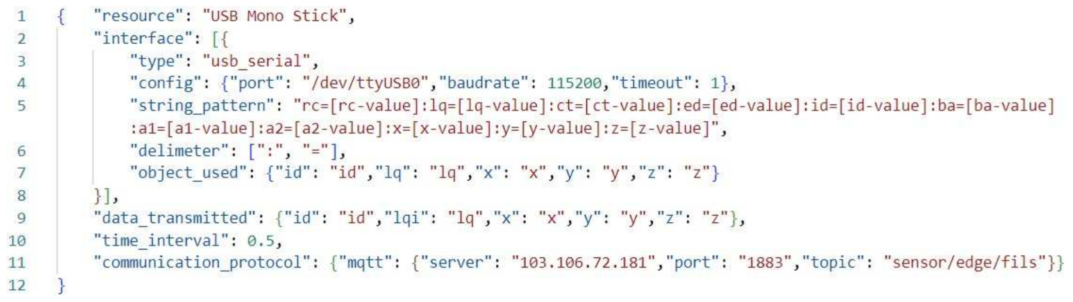
adaptability and the validity of the edge device framework in *SEMAR*. Table 4.1 presents the device and software specifications for this evaluation.

Table 4.1: Device and software specifications of *FILS15.4*.

Components	Items	Specifications
Edge Device	Model Operating System	Raspberry Pi 4B Linux Raspbian
Sensor Device	Model Sensor Interface Communication Method Collected Data	Twelite Mono Stick USB Serial Communication *id, LQI, accelerometer x, y and z*

I evaluated the ability of the edge device framework to automatically install the edge configuration built on *SEMAR* to the edge device, collect sensor data, convert them, and send them to the server by following the configuration file. In addition, I evaluated the configuration update feature by modifying the edge configuration setting and remotely deploying it to the edge device through the user interface of *SEMAR*.

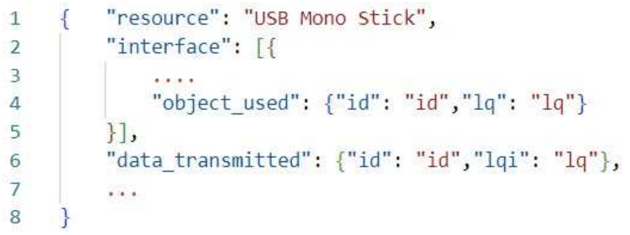
Figure 4.7 shows the initial configuration file for *FILS15.4*. The interface includes the configuration of the serial communication for collecting data from a USB receiver and the parameter for obtaining the necessary data by converting them to the standard format. According to the edge configuration, the system sends sensor data that consist of *ID*, *LQI*, and *accelerometer* x, y, z, to the server at every 0.5 s (500 ms) through the *MQTT* communication.



```
{   "resource": "USB Mono Stick",
    "interface": [{
        "type": "usb_serial",
        "config": {"port": "/dev/ttyUSB0","baudrate": 115200,"timeout": 1},
        "string_pattern": "rc=[rc-value]:lq=[lq-value]:ct=[ct-value]:ed=[ed-value]:id=[id-value]:ba=[ba-value]:a1=[a1-value]:a2=[a2-value]:x=[x-value]:y=[y-value]:z=[z-value]",
        "delimeter": [":", "="],
        "object_used": {"id": "id","lq": "lq","x": "x","y": "y","z": "z"}
    }],
    "data_transmitted": {"id": "id","lqi": "lq","x": "x","y": "y","z": "z"},
    "time_interval": 0.5,
    "communication_protocol": {"mqtt": {"server": "103.106.72.181","port": "1883","topic": "sensor/edge/fils"}}
}
```

Figure 4.7: Edge configuration for receiver device of *FILS15.4*.

Figure 4.8 illustrates the updated edge configuration for *FILS15.4*. It is changed from the initial configuration. The configuration was modified by removing the accelerometer data from the result of the data converter process, and only transmitting *ID* and *LQI* data to the server. The data transmission interval is similar to the previous configuration.



```
{   "resource": "USB Mono Stick",
    "interface": [{
        ....
        "object_used": {"id": "id","lq": "lq"}
    }],
    "data_transmitted": {"id": "id","lqi": "lq"},
    ...
}
```

Figure 4.8: Updated edge configuration for receiver device of *FILS15.4*.

Figure 4.9 shows the data visualization of *FILS15.4* through the *SEMAR* user interface. The initial configuration part indicates that the edge device can collect data from the USB receiver, convert them, and send them to the server by following the initial configuration in Figure 4.7. The updated configuration part represents the edge device when it collects, processes, and transmits data by following the updated configuration in Figure 4.8.

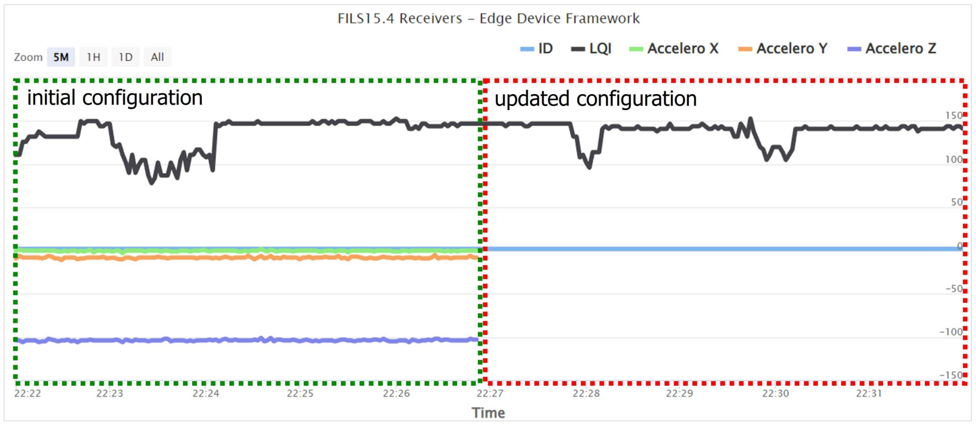


Figure 4.9: Data visualization of the *FILS15.4* receiver device.

4.7 Application for Data Logging System

As the second application, the data logging system is integrated to enable real-time monitoring of the temperature data of some materials during the quenching heat treatment process.

4.7.1 System Overview

Figure 4.10 illustrates the overview of the *data logging system* architecture. This system uses *midi Logger GL240* with *WLAN B-568* that is provided by *Graphtec* [82] to capture the temperature data during the quenching heat treatment process by attaching the sensor to the material. The treatment process is used for hardening steel by putting the material into the heater machine to improve metal performances. *WLAN B-568* provides the *HTML* web service for displaying the data collected by the data logger. The integration of the data logger with the IoT application server platform is as follows:

- The edge device for the data logging system captures raw sensor data in the HTML format by accessing the data logger web services through wireless communications;
- It reads the input HTML data, extracts the temperature value using web scraping techniques, and transforms it into JSON format;
- It transmits the JSON data to the *SEMAR* platform through the *MQTT* communication protocol;
- The *SEMAR* platform receives, processes, and saves the sensor data in the database;
- The *SEMAR* platform displays the sensor data through the user interfaces.

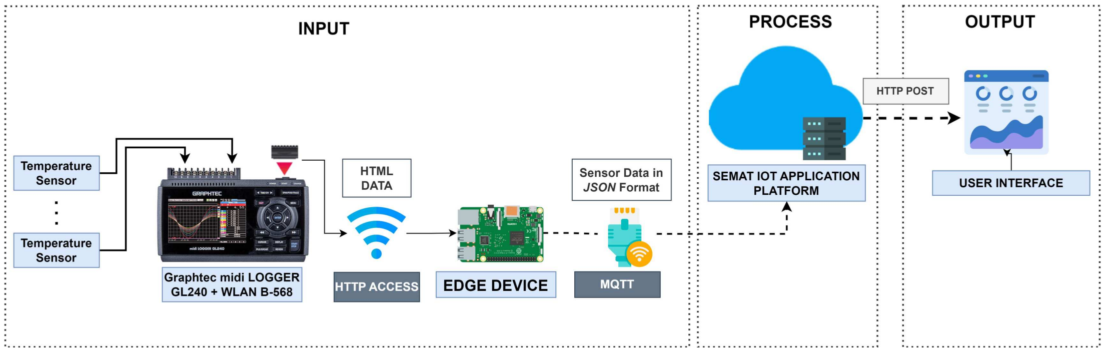


Figure 4.10: System overview of *data logging system*.

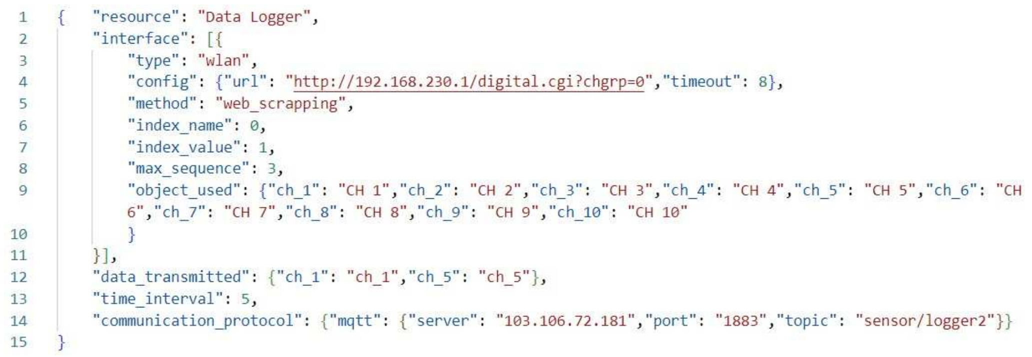
4.7.2 Evaluation of Implementation

I evaluated the implementation of the edge device framework for the *data logging system* by running it in the #1 Engineering Building at Okayama University. The evaluations were intended to verify the adaptability and the validity of the edge device framework in *SEMAR*. Table 4.3 presents the device and software specifications for this evaluation.

Table 4.3: Device and software specifications for *data logging system*.

Components	**Items**	**Specifications**
Edge Device	Model	Raspberry Pi 4B
	Operating System	Linux Raspbian
Sensor Device	Model	midi Logger GL240 and WLAN B-568
	Sensor Interface	Wireless Connection
	Wireless LAN Mode	Access Point
	Wireless LAN IP	*192.168.230.1*
	Web Services URL	`http://192.168.230.1/digital.cgi?chgrp=0`
	Communication Method	*HTTP* Communication
	Collected Data	*temperature*

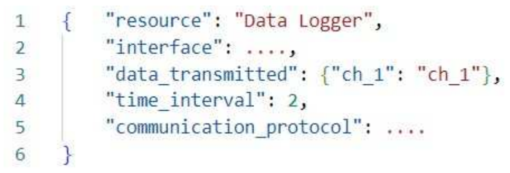
Figure 4.11 shows the initial configuration file for the *data logging system*. The edge configuration indicates that the edge device collects data from the data logger through the wireless network. It transmits the measured temperature data from *channels* 1 and 5 to the server every 5*s* through the *MQTT* communication.



```
{   "resource": "Data Logger",
    "interface": [{
        "type": "wlan",
        "config": {"url": "http://192.168.230.1/digital.cgi?chgrp=0","timeout": 8},
        "method": "web_scrapping",
        "index_name": 0,
        "index_value": 1,
        "max_sequence": 3,
        "object_used": {"ch_1": "CH 1","ch_2": "CH 2","ch_3": "CH 3","ch_4": "CH 4","ch_5": "CH 5","ch_6": "CH
        6","ch_7": "CH 7","ch_8": "CH 8","ch_9": "CH 9","ch_10": "CH 10"
        }
    }],
    "data_transmitted": {"ch_1": "ch_1","ch_5": "ch_5"},
    "time_interval": 5,
    "communication_protocol": {"mqtt": {"server": "103.106.72.181","port": "1883","topic": "sensor/logger2"}}
}
```

Figure 4.11: Edge configuration for edge device in data logging system.

Figure 4.12 shows the updated edge configuration of the data logger monitoring system. It was modified from the initial configuration. In this configuration, the transmitted data were changed by only sending the temperature data from *channel*1 every 2*s* through the *MQTT* communication.



```
{   "resource": "Data Logger",
    "interface": ....,
    "data_transmitted": {"ch_1": "ch_1"},
    "time_interval": 2,
    "communication_protocol": ....
}
```

Figure 4.12: Updated edge configuration for edge device of data logging system.

Figure 4.13 illustrates the data visualization of the data logging system. The initial configuration part represents the edge device for collecting, processing, and transmitting data according to the initial configuration in Figure 4.11. Additionally, the updated configuration part shows the data sent by the edge device when the configuration is modified according to Figure 4.12.

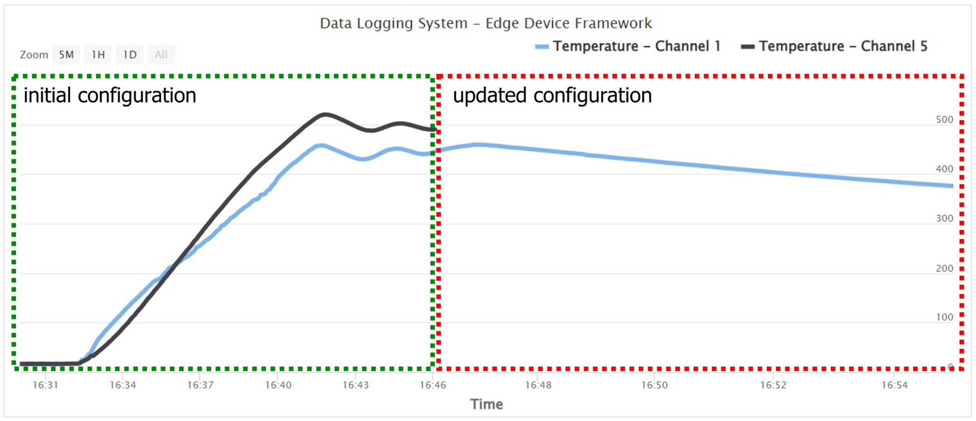


Figure 4.13: Data visualization of data logging system.

4.8 Evaluations of Edge Device Framework

In this section, I evaluated the implementation of the *edge device framework*.

4.8.1 Performance of Main Service

The first evaluation of the edge device framework's performance involved investigating the average CPU and memory usage of the main service program while collecting and transmitting sensor data at various time intervals. This evaluation was crucial for assessing the computational performance of the framework during the main phase. To carry out this evaluation, I employed the data logging system application and measured the average memory and CPU usage during the experiment time as shown in Figures 4.14 and 4.15. I tested different time intervals ranging from $0.1s$ to $10s$ for three minutes each and utilized the feature described in Section 4.4 to modify the time interval configuration.

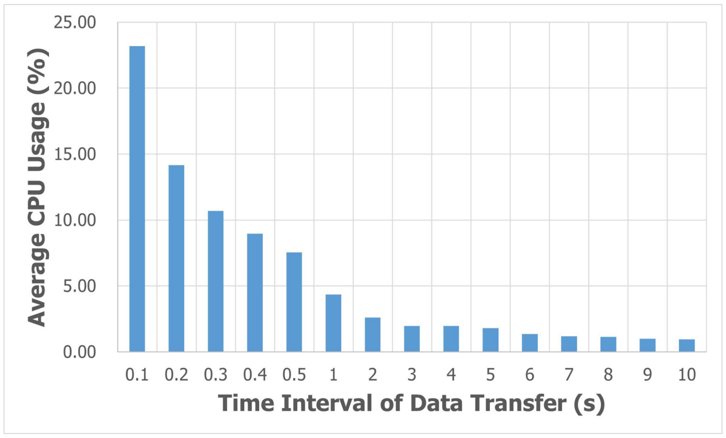


Figure 4.14: Average CPU usage rate of main services with different time intervals.

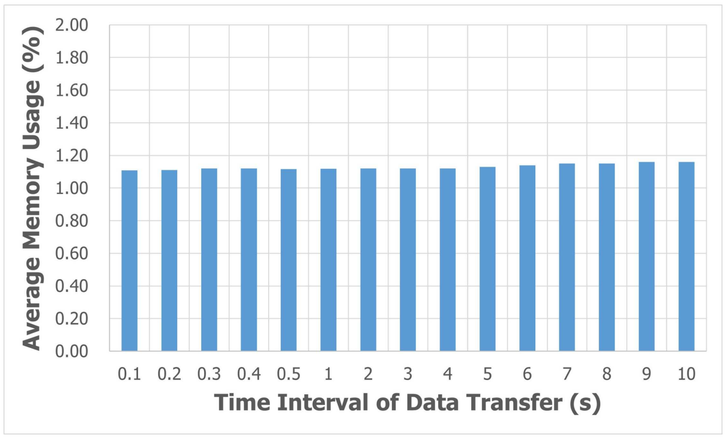


Figure 4.15: Average memory usage of main services with different time intervals.

The results indicate that shorter time intervals require higher percentages of the CPU usage, where all the experimental results fall below 25%. Moreover, the amount of the memory usage remains relatively stable across time intervals, suggesting that the proposed system operates without demanding excessive computational resources.

The second evaluation involved examining the average response time of the web services when accessed by multiple users simultaneously via *HTTP POST* communications. The edge device framework was installed on a *Raspberry Pi*, and a considerable amount of sensor data were stored. To simulate multiple users, I developed a simulation program that generates virtual users, and ran it on personal computers connected to the *Raspberry Pi* via Ethernet in the local area network. During the experiments, I increased the number of user accesses from 5 to 150, with each virtual user representing an actual user or system using the device data. All the virtual users used similar parameter requests to access sensor data stored in local data storage.

To measure the response time, I calculated the time difference between the case where a virtual user sends a request to the web services and the case where it receives the response message. The response message is the $56KB$ JSON message containing 500 records of data. During the experiment, I also evaluated the throughput of web services, which was $2.3MB/s$. It can handle 41 requests per second.

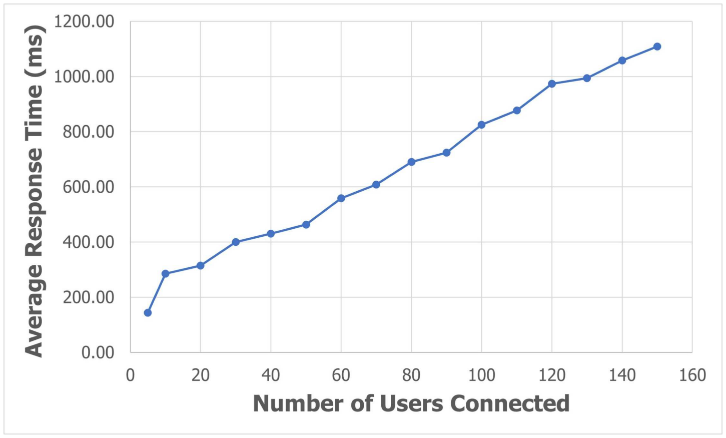


Figure 4.16: Average response time of web services with different numbers of users connected.

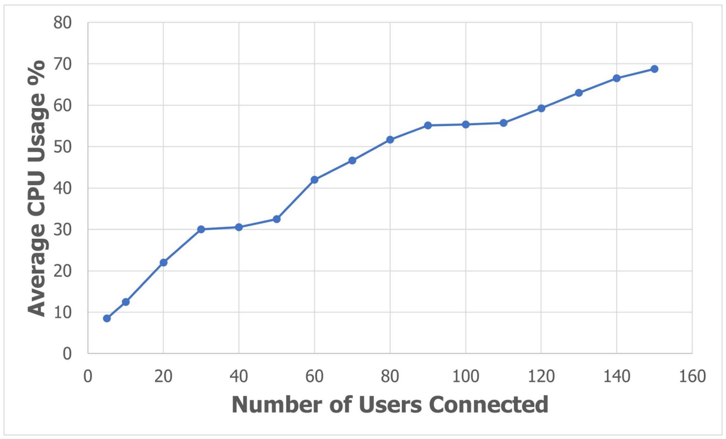


Figure 4.17: Average CPU usage rate of web services with different numbers of users connected.

Figures 4.16 and 4.17 illustrate the average response time and the CPU usage rate when the number of virtual users increases from 5 to 150. The average response time is 824*ms*, and the CPU usage rate is 55% for 100 devices with the response message containing 500 data records. These results indicate that the proposed edge device framework can accommodate hundreds of users with a reasonable response time and the CPU usage rate.

4.8.2 Performance of Filtering Functions

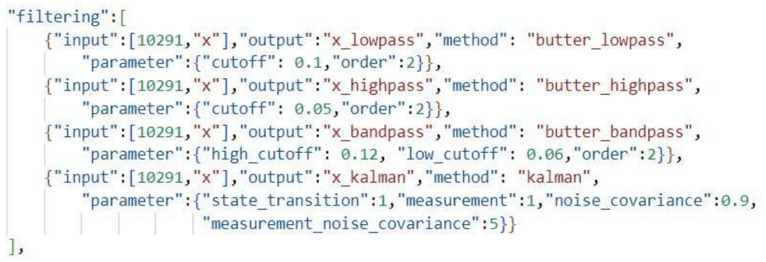
I evaluated the implementation of the digital filter functions in the edge device framework. For this purpose, the *FILS15.4* IoT application system is used as the target.

4.8.2.1 Experimental Scenario

The experiments were conducted in the #2 Engineering Building of Okayama University. I installed the edge device framework in the *Raspberry Pi*, which served as the receiver device in the *FILS15.4* system. It was connected to *Mono Stick* to collect *ID*, *LQI*, and *accelerometer x*, *y*, *z* data periodically every 500*ms* through USB serial communication. In order to generate data for moving user scenarios, a user was equipped with a transmitter and moved in the same room as the receiver during experiments.

4.8.2.2 Digital Filter Results

I investigated all the digital filtering techniques implemented in the framework. The experiment was conducted in two parts. First, I evaluate *Butterworth* and *Kalman* filters for filtering the *accelerometer x* data simultaneously.



```
"filtering":[
    {"input":[10291,"x"],"output":"x_lowpass","method": "butter_lowpass",
        "parameter":{"cutoff": 0.1,"order":2}},
    {"input":[10291,"x"],"output":"x_highpass","method": "butter_highpass",
        "parameter":{"cutoff": 0.05,"order":2}},
    {"input":[10291,"x"],"output":"x_bandpass","method": "butter_bandpass",
        "parameter":{"high_cutoff": 0.12, "low_cutoff": 0.06,"order":2}},
    {"input":[10291,"x"],"output":"x_kalman","method": "kalman",
        "parameter":{"state_transition":1,"measurement":1,"noise_covariance":0.9,
                    "measurement_noise_covariance":5}}
],
```

Figure 4.18: Edge configuration for filtering *accelerometer x* data.

Figure 4.18 shows the configuration file in this experiment. For the *Butterworth* filters, I designed second-order filters with different cutoff frequencies in *Hz*. For the *Kalman* filter, I specified the process noise covariance of 0.9 and the measurement noise covariance of 5. Figure 4.19 illustrates the sensor and the filtered data of the *accelerometer x* by following the configuration in Figure 4.18.

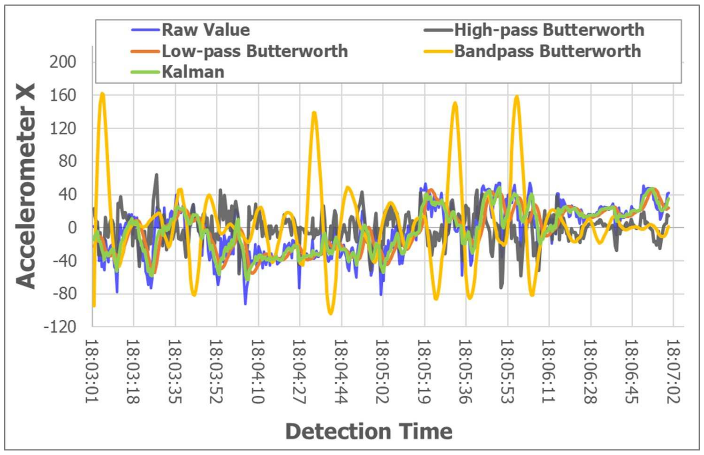
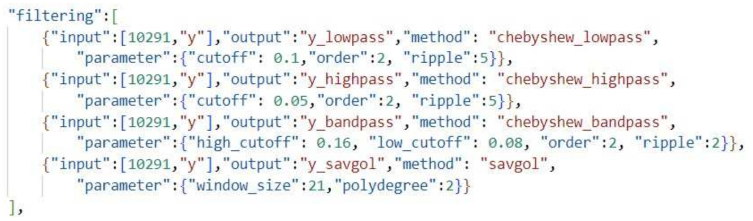


Figure 4.19: Results of *Butterworth* and *Kalman* filters.

Next, I evaluate *Chebyshev-I* and *Savitzky-Golay* filters for filtering the *accelerometer y* data simultaneously. Figure 4.20 shows the configuration file utilized in this experiment. For the *Chebyshev-I* filters, the second-order filters were designed with the ripple parameter of 5 and different cutoff frequencies for each technique. For the *Savitzky-Golay* filter, I defined parameters consisting of the polynomial degree of 2 and the window size of 21.



```
"filtering":[
    {"input":[10291,"y"],"output":"y_lowpass","method": "chebyshew_lowpass",
        "parameter":{"cutoff": 0.1,"order":2, "ripple":5}},
    {"input":[10291,"y"],"output":"y_highpass","method": "chebyshew_highpass",
        "parameter":{"cutoff": 0.05,"order":2, "ripple":5}},
    {"input":[10291,"y"],"output":"y_bandpass","method": "chebyshew_bandpass",
        "parameter":{"high_cutoff": 0.16, "low_cutoff": 0.08, "order":2, "ripple":2}},
    {"input":[10291,"y"],"output":"y_savgol","method": "savgol",
        "parameter":{"window_size":21,"polydegree":2}}
],
```

Figure 4.20: Edge configuration for filtering *accelerometer y* data.

Figure 4.21 illustrates the sensor and the filtered data of the *accelerometer y* by following the configuration in Figure 4.20. The experimental results obtained for the digital filtering functions illustrate that digital signal processing techniques have been successfully implemented for real-time data filtering. These functions are able to reduce noise and enhance the reliability of the collected sensor data. These findings indicate that the selection of suitable filtering techniques and the definition of their parameters are important to achieve better filtering performance. Furthermore, the results show that the framework provides users with the capability to flexibly customize the filter settings based on their specific requirements through the edge configuration file.

4.8.2.3 Cascading Filter Results

Finally, I evaluate the implementation of cascading filter functions in the framework. Figure 4.22 shows the configuration file utilized in this experiment. I designed the cascading filter by combining the low-pass Butterworth filter and the Kalman filter to filter the *accelerometer z* data. I

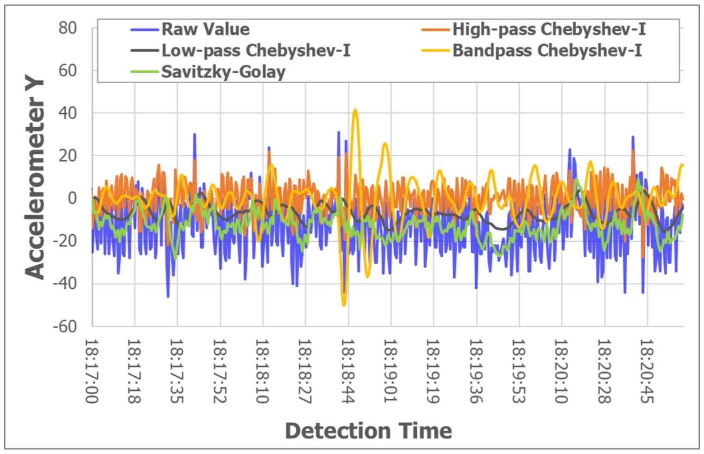


Figure 4.21: Results of Chevyshev-I and Savitzky-Golay filters.

used second-order filters with a cutoff frequency of 0.15 Hz for the low-pass Butterworth filter. While I specified the process noise covariance of 0.5 and the measurement noise covariance of 1 the design of the Kalman filter. To investigate the effectiveness of the cascading filter, the digital filter functions of the low-pass Butterworth and Kalman filters were also added.

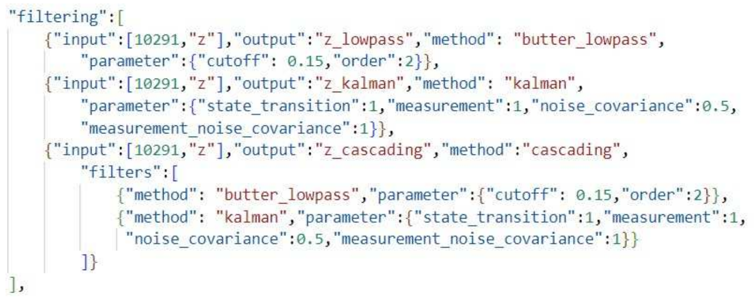

```
"filtering":[
    {"input":[10291,"z"],"output":"z_lowpass","method": "butter_lowpass",
        "parameter":{"cutoff": 0.15,"order":2}},
    {"input":[10291,"z"],"output":"z_kalman","method": "kalman",
        "parameter":{"state_transition":1,"measurement":1,"noise_covariance":0.5,
        "measurement_noise_covariance":1}},
    {"input":[10291,"z"],"output":"z_cascading","method":"cascading",
        "filters":[
            {"method": "butter_lowpass","parameter":{"cutoff": 0.15,"order":2}},
            {"method": "kalman","parameter":{"state_transition":1,"measurement":1,
             "noise_covariance":0.5,"measurement_noise_covariance":1}}
        ]}
],
```

Figure 4.22: Edge configuration for filtering *accelerometer z* data.

Figure 4.23 show illustrates the sensor and the filtered data of the *accelerometer z* by following the configuration in Figure 4.22.

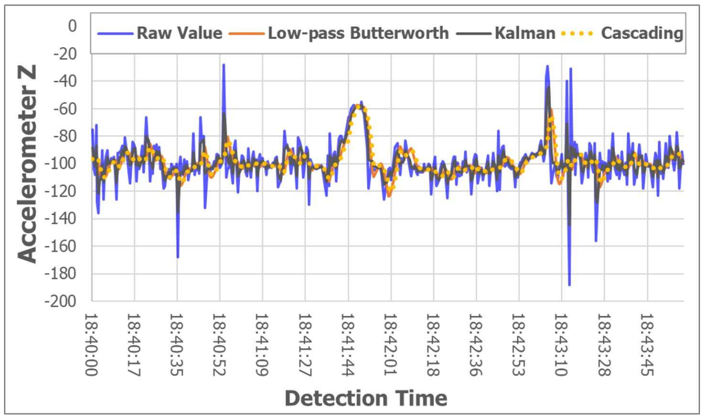


Figure 4.23: Results of Cascading filter.

The experiment results illustrate that the cascading filter functions have been successfully implemented. These allow the combination of multiple filters to perform more advanced filtering operations.

4.8.3 Comparative Analysis

To illustrate the latest developments in edge computing frameworks, I evaluated several comparable models and extracted relevant information from their published papers. I compared features of the edge device framework with eight research works taking similar approaches in the literature. I compiled a list of features to be considered for comparing different edge computing systems frameworks. They were used to characterize each proposal and included the following:

- The *main purpose* was to identify the issue that the proposed system intends to address and the key reason for selecting it to run edge IoT applications.
- *Edge devices* represent devices that installed an edge computing framework system.
- *Dynamic deployment* shows the ability to allow users to dynamically configure the flow system to run their own edge applications based on hardware and process requirements (Yes or No).
- *Remotely update* indicates the capability to remotely update the system (Yes or No).
- *Data conversion* implies the capability to preprocess data across several devices into a standard format (Yes or No).
- *Scalability* demonstrates the ability to expand their applications and to execute the number of data processing requests simultaneously (Yes or No).
- *Interoperability* indicates the capability to connect through several widely adopted and supported protocols provided by multiple devices (Yes or No).
- *Cross-vendor capabilities* illustrate the capacity of edge computing to collaborate with multiple vendors to develop complex IoT application platforms (Yes or No).

Table 4.5 compares the fulfillment of the eight features among the eight related works and our proposed edge devices framework.

Overview

Sajjad et al. in [83], Banerjee et al. in [84], and Ullah et al. in [87] developed systems that are consistent with the main objective of the edge computing framework by collecting data from diverse devices. Moreover, Rong et al. in [88] and Berta et al. in [5] created an edge computing framework that can gather data and connect to the actuator as the system output, which is similar to our edge device framework. Our framework is a general framework for edge computing and has the ability to connect with several IoT networks and to offer multiple output components that utilize the acquired data.

Edge Devices

Multiple works have used personal computers for installing and operating the frameworks. Nevertheless, they do not support the GPIO connectivity that is commonly used in sensor devices. Chen et al. in [85] and Berta et al. in [5] have implemented framework systems using single-board computer devices, such as the *Raspberry Pi*, which has significant benefits. Hence, I chose to deploy the proposed framework on these devices. This approach enables sensors to connect directly to the single-board computer devices for data collections, making the development of IoT application systems more straightforward.

Table 4.5: The comparative evaluation between the proposed framework and the existing related studies.

Work Reference	**Main Purpose**	**Edge Devices**	**Dynamic Deployment**	**Remotely Update**	**Data Conversion**	**Scalability**	**Interoperability**	**Cross-vendor Capabilities**
[83]	Data stream processing and task management	Wi-Fi Home Gateway	✓	✗	✗	✓	✓	✗
[84]	Edge devices gateways and support tool	Personal Computer and Server	✓	✓	✗	✓	✓	✓
[85]	Edge devices for smart manufacturing	Single-Board Computer	✓	✓	✓	✓	✗	
[86]	Edge framework for smart farming	Personal Computer	✓	✗	✗	✓	✓	✗
[87]	Edge computing gateways	Server	✓	✗	✗	✓	✓	✓
[88]	Edge computing framework	Personal Computer	✓	✓	✓	✓	✓	✗
[89]	Edge devices for smart home	Personal Computer	✓	✗	✗	✓	✓	✗
[5]	Edge computing framework	Single-Board Computer	✓	✓	✗	✓	✓	✗
Our Proposal	General edge computing framework	Single-Board Computer	✓	✓	✓	✓	✓	✓

Framework Features

In terms of framework features, all the related works offer capabilities for gathering data from IoT devices and sending them to a cloud server. However, as in our proposal, the works by Chen et al. in [85] and Rong et al. in [88] included the feature to process sensor data by converting them based on user-defined configurations.

All the works examined provided the capability to dynamically set up and deploy the framework using the connected devices as the main requirement. Some works required direct access to the devices for operations. Notably, Banerjee et al. [84], Chen et al. [85], Rong et al. [88], Berta et al. [5], and our proposed framework allow users to remotely update the configuration from the cloud server.

Scalability, Interoperability, and Cross-Vendor Capabilities

All the works that have been reviewed focus on incorporating the scalability and interoperability in the functionality. However, some of them have the limited methods of connectivity for linking IoT devices to the edge framework. For instance, Sajjad et al.'s work [83] only allows the connectivity via Wi-Fi communications, whereas Zamora et al. [86] and Sharif et al.'s works [89] only permit connections from control unit devices to receive sensor data. Some works consider the cross-vendor capabilities of edge computing frameworks, particularly regarding data output components. Banerjee et al. [84], Ullah et al. [87], and the proposed framework allow the user to access to sensor data from edge devices using the *REST API*. However, only the proposed framework provides the additional features that allow data transmissions to various cloud computing vendors through *MQTT* and *HTTP POST* communications.

4.9 Summary

This chapter presented the design and implementation of the *Edge Device Framework* in the *SEMAR* IoT Application Server platform. It allows users to remotely optimize device utilizations by configuring it through the *SEMAR* interface. The framework defines the connectivity of sensor interfaces, filtering functions, transmitted sensor elements, communication protocol, local data storage, local visualization, and data transmission interval on the server. The framework was applied to two IoT application systems. The evaluation results verified the adaptability and validity of the proposed framework.

Chapter 5

Study of AI Techniques Integration with Use Cases in SEMAR

This chapter provides an overview of current AI techniques and their use cases in IoT applications. They include predictive analytics, image classification, object recognition, text spotting, auditory perception, *natural language processing (NLP)*, and collaborative AI. The key characteristics of these techniques are described, identifying the critical parameters for integrations. Based on these findings, I design a seamless integration of AI capabilities into the *SEMAR* platform. In addition, several IoT use cases are discussed to demonstrate how SEMAR can support their development.

5.1 Literature Review on Use Cases of AI Techniques in IoT Applications

In this section, I present a review of use cases of AI techniques in literature for IoT applications as comprehensively as possible.

5.1.1 Methodology

The main purpose of this literature review section is to identify AI techniques that have been frequently used in IoT applications, including algorithms, characteristics, and how they can be implemented in IoT application use cases. To achieve this, I followed a structured research methodology. It consists of identifying the trends of applied AI techniques, finding the related literature, investigating characteristics, and analyzing necessary requirements for seamless integrations.

First, I identified the trends of applied AI techniques in the development of IoT application systems. For this purpose, I explored the surveyed papers that discuss applied AI techniques in IoT application systems with their potential. According to the findings in several studies [90–94], I selected predictive analytics, image classification, object detection, text spotting, auditory perception, NLP, and collaborative AI as the typical AI techniques to be explored in this thesis.

In the next step, I systematically selected relevant papers for reviews from popular scientific databases such as *Scopus*, *Elsevier*, and *IEEE*. To capture the current state of each AI technique, the literature review was limited to publications that were published between 2019 and 2023. These publications were selected based on a combination of keywords representing each AI technique identified in the previous step, as well as the domains of IoT application use cases. They included smart environments, smart manufacturing, smart cities, smart homes, smart buildings, smart healthcare, smart agriculture, smart farming, and smart laboratories.

To investigate the characteristics of each technique and its application use cases, I considered critical features such as software requirements, *I/O* data types, processing methods, and computations. Finally, I analyzed the unique strengths and requirements of each AI technique with the specific purpose of designing seamless integration. Following this insight, I designed the AI integration into the *SEMAR* platform.

5.1.2 Predictive Analytics

This subsection provides an overview of the current state of the art for integrating predictive analytics into IoT systems by reviewing papers with considering applications use cases.

5.1.2.1 Introduction

The integration of AI into an IoT application has changed the way how data is collected, processed, and visualized. As an IoT application requires the ability to rapidly extract meaningful information from data, a function or a system that enables the identification of data patterns and trends in a real-time manner becomes a critical issue. Predictive analytics is one of the AI techniques often used to solve this issue. It finds knowledge in current and past data to generate predictions of future events by using machine learning, statistics, and data mining techniques [95].

In the context of IoT, predictive analytics analyzes historical sensor data saved in the database to predict future events or data trends. It is often used to perform anomaly detection, predictive maintenance, optimization, and decision-making in a real-time or near-real-time manner.

5.1.2.2 Use Cases in IoT Applications and Characteristics Overview

Several papers discussed use cases that provide the potential for using predictive analytics techniques to improve IoT application systems.

Forecasting future environmental conditions based on historical data that were collected by sensors is one of the goals of smart environments. As a direction toward this goal, in [96], Imran et al. proposed an IoT-based simulation system that predicts fire spread and burned areas in mountainous areas. In [97], Hussain et al. used predictive analytics techniques to forecast the level of *carbon monoxide (CO)* concentration in the area around a garbage bin. Jin et al. in [98] proposed a novel approach for predicting *particulate matter (PM)* 2.5 concentrations using a *Bayesian* network. This study introduced a *Bayesian*-based algorithm that provides a potential robust prediction for time-series data.

Bampoula et al. in [99] and Teoh et al. in [100] illustrate the effectiveness of integrating predictive analytics into IoT application systems to improve industrial asset management by accurately estimating machine or equipment conditions.

In [101], Shorfuzzaman et al. presented the practical implementation for minimizing energy consumption of home appliances in a smart home context using predictive analytics. Then, Guo et al. in [102] provided a system for predicting building electricity consumption based on a small-scale data set collected by sensors.

Nancy et al. in [103] and Subahi et al. in [104] introduced the application of predictive analytics techniques into IoT cloud-based systems to forecast disease conditions based on medical data of the patients in the context of smart healthcare.

In [105], Patrizi et al. demonstrated the implementation of a virtual-based soil moisture sensor in the context of smart farming applications. This can be achieved by estimating soil moisture using collected sensor data using predictive analytics techniques. Kocian et al. in [106] introduced an IoT system for smart agriculture.

Table 5.1: Key characteristics of predictive analytics technique in current studies.

Ref.	**Algorithms**	**Software Requirements**	**Data Types**		**Processing Methods**	**Computations**
			Input	**Output**		
[96]	ANN with PCR and Kalman filter	Python	Time-series data	Predicted area	Filtering and real-time data processing	Raspberry PI with 3.00 GB RAM
[97]	LSTM	Python, TensorFlow	Time-series data	Predicted CO level	Real-time data processing	Google Cloud Server
[98]	Bayesian Network	Python	Time-series data	Predicted PM2.5 level	Missing data handling, data normalization, and data correlations	AMD R7-5800 processor 4.0 GHz with 16GB of RAM
[99]	LSTM	Python, TensorFlow	Time-series data	Predicted machine states	Data transformation and real-time data processing	Intel CoreTM i7 CPU with 8.00 GB RAM
[100]	Logistic Regression	Azure Machine Learning *REST API* services	Time-series data	Predicted equipment health states	Real-time data processing	Azure Machine Learning
[101]	LSTM and ARIMA	Python, TensorFlow	Time-series data	Predicted energy consumption	Missing data handling, outlier detection, data transformation	Intel CoreTM i7 CPU with 8.00 GB RAM
[102]	ARIMA and SVR	Python	Time-series data	Predicted electric consumption	Missing data handling, data normalization, and data correlations	Intel Core i5 CPU with 8.00 GB RAM
[103]	Bidirectional LSTM	Python, TensorFlow	Time-series data	Predicted diagnosis of heart disease	Data Filtering	i2k2 Cloud platform
[104]	Self-Adaptive Bayesian	-	Time-series data	Predicted diagnosis of heart disease	Data normalization	-
[105]	LSTM	Python, TensorFlow	Time-series data	Predicted soil moisture	Data correlations and data synchronization	-
[106]	Dynamic Bayesian	MATLAB™	Time-series data	Predicted ET value	Real-time data processing	-

Table 5.1 summarizes the characteristics of predictive analytics techniques that were discussed in this subsection.

Long Short-Term Memory (LSTM) is the widely used algorithm for predicting future events. It belongs to the variant of *Recurrent Neural Network (RNN)* architecture, which effectively learns and retains information over a long period using cell states [107]. The works by Shorfuzzaman et al. in [101] and Guo et al. in [102] utilized the capabilities of the *Autoregressive Integrated Moving Average (ARIMA)* algorithm [108] to construct robust data models for predicting future values in time series data. Taking it one step further, Guoh et al. in [102] combined *Support Vector Regression (SVR)* with *ARIMA* to predict energy consumption. Then, Imran et al. in [96] integrated *Principal Component Regression (PCR)* and *Artificial Neural Network (ANN)* for an effective predictive model in their application system.

In addition, the works by Kocian et al. in [106], Jin et al. in [98], and Subahi et al. in [104] used *Bayesian*-based approaches. While *ARIMA* focuses on constructing models of the time-series data, *Bayesian* approaches take a different approach by generating prior knowledge in the form of a probability distribution for predicting the data. This prior knowledge represents initial beliefs. It is then updated with observed data using *Bayes' theorem*, allowing for more flexible and robust modeling. Moreover, *Bayesian* approaches can be implemented in scenarios with limited datasets, as mentioned in the work by Kocian et al. in [106].

Predictive analytics is essential to estimate future values or labels in real-time scenarios of IoT applications. The effective implementation of this technique requires consideration of several key elements. First, a database system that can handle time-series data is critical for efficient data storage and retrieval. Second, the pre-processing capabilities must be implemented to prevent potential error data in the collected data and improve reliability. Third, the IoT system must have real-time data processing capabilities to perform immediate analysis for rapid forecasting and decision-making. Fourth, *Python*, with its extensive support for algorithms for predictive analytics such as *LSTM*, *ARIMA* and *Bayesian*, becomes the suitable option for the software environment. As shown in Table 5.1, the predictive analytics can be deployed on either servers or edge devices such as *Raspberry Pi*.

5.1.3 Image Classification

In this subsection, I review papers emphasizing IoT application use cases for image classifications.

5.1.3.1 Introduction

Computer vision is the field of AI that mimics human intelligence to understand image data. It enables machines to see and recognize objects from visual images to facilitate decision-making [109]. Techniques such as image classification and object detection are part of the fields in computer vision, where image classification refers to the ability to identify categories of images.

In the IoT domain, image classification plays an important role in recognizing visual data using a classification model. Typically, the data model is trained using labeled image datasets, where each image is assigned to a specific category.

5.1.3.2 Use Cases in IoT Applications and Characteristics Overview

The implementation of an image classification in an IoT application has been explored in numerous papers. Each paper demonstrated the effectiveness of image classification algorithms in addressing vision-based use cases in a variety of applications.

For IoT applications in agriculture, image classification is a valuable technique for monitoring crops and detecting plant diseases. In [110], Chouhan et al. introduced a system for detecting galls, a plant disease that affects leaves, using captured images. In a separate study in [111], Munawar et al. showed that this technique is also suitable for drone-based IoT applications in environmental monitoring systems.

Image classification is a proven AI technique for supporting diagnostic processes through visual data analysis. Abd Elaziz et al. in [112] and Saleh et al. in [113] demonstrated the effectiveness of image classifications in improving diagnostic capabilities within smart healthcare use cases. In [113], Saleh et al. employed a hybrid approach of combining *Convolutional Neural Network (CNN)* and *Support Vector Machine (SVM)* to classify lung cancers based on *computed tomography (CT)* scan images.

In addition, Iyer et al. in [114] and Medus et al. in [115] demonstrated the versatility of image classifications and extended its benefits to diverse areas beyond healthcare, such as transportation infrastructure and quality control in food production.

Table 5.2: Key characteristics of image classification techniques.

Ref.	**Algorithms**	**Software Requirements**	**Data Types**		**Processing Methods**	**Computations**
			Input	**Output**		
[110]	FBFN	Python and OpenCV	Captured images	Leaf gall detection (Boolean)	Image pre-processing, feature extraction, hyperparameters optimization, and real-time data processing	-
[111]	CNN	Python, OpenCV, and TensorFlow	Captured images	Flooded detection (Boolean)	Image pre-processing	Intel Core i7 CPU
[112]	Deep learning (MobileNetV2 and DenseNet169)	Python, OpenCV, and TensorFlow	Medical images	Medical diagnostic classes	Feature extraction, feature selection, and *REST API* services	-
[113]	The hybrid of CNN and SVM	Python, OpenCV, and TensorFlow	Medical images	Lung cancer classes	Hyperparameters optimization	Intel Core i5 CPU with 16.00 GB of RAM and NVIDIA GeForce RTX 2060 GPU
[114]	CNN	Python, OpenCV, and TensorFlow	Captured images	Fracture detection (Boolean)	Image pre-processing and feature extraction	Raspberry Pi 3
[115]	CNN	Python, OpenCV, TensorFlow, and Keras	Captured images	Failure detection (Boolean)	Hyperparameters optimization	Intel Core i7 CPU with 8.00 GB of RAM

Table 5.2 shows the overview of the characteristics of image classification techniques in the literature discussed in this chapter.

The CNN algorithm [115] has been widely used for image classifications in various applications. The architecture of the CNN algorithm allows it to be integrated with other algorithms, such as SVM. In [113], Saleh et al. presented a hybrid algorithm with CNN and SVM to achieve robust performance. SVM is used to generate the classification result using features extracted by CNN. This approach leverages the strengths of both models and enhances accuracy in classification tasks.

Several researchers have proposed alternative approaches to classifying images instead of the CNN algorithm. Abd Elaziz et al. in [112] have developed a deep learning model that combines *MobileNet* and *DenseNet* architectures to extract medical image representation. This model is able to extract complex features from medical images, making it useful for better understanding and diagnosing medical conditions. Medus et al. in [115] presented the implementation of the *Fuzzy-Based Functional Network (FBFN)* algorithm that integrates fuzzy logic with function network capabilities. This approach allows the user to apply the image classification process in real-time.

There are several key elements to be considered for implementing image classification algorithms. As the programming language, Python is often used. Then, libraries such as *TensorFlow*, *Keras*, and *OpenCV* are installed to implement various deep learning-based image processing algorithms. Since the input data includes image files, storage capacity becomes necessary. To achieve high performance, additional functions such as noise reductions in images are applied. Hyperparameter optimization is also applied to optimize the performance of the model under different input data. Finally, for the computation device, researchers often use GPU-integrated and memory-optimized approaches. For instance, Saleh et al. in [113] improved the performance by adding GPUs, accelerating the training phase, and reducing the processing time during the detection phase.

5.1.4 Object Detection

In this subsection, I provide an overview of integrating object detection techniques into IoT applications.

5.1.4.1 Introduction

Object detection is one of the successful AI techniques in the field of computer vision. While image classification focuses on categorizing entire images, object detection goes a step further by recognizing both the categories and the precise locations of specific objects within the images. This technique is often used as the first step to perform other tasks, including recognizing faces, estimating poses, and analyzing human activity.

In the context of IoT, object detection plays a critical role in various applications, such as autonomous video surveillance, smart cities, and manufacturing. Its integration into IoT devices facilitates real-time analysis of video sequences, which is essential for ensuring safety and efficiency in various environments. However, this integration brings new challenges in detecting moving objects and rapidly extracting their features. Addressing these challenges requires consideration of computational resources to efficiently manage the huge amount of IoT data, especially in use cases of intelligent surveillance systems.

5.1.4.2 Use Cases in IoT Applications and Characteristics Overview

In this section, I explore the papers that discuss the integration of object detection in the IoT domain. They presented how object detection algorithms are applied in various application scenarios.

In the context of smart cities, object detection helps to improve urban management. It allows the detection and localization of various entities in urban environments, such as vehicles, pedestrians, and objects, for intelligent transportation, intelligent surveillance, and drone monitoring. Zhou et al. in [116], Abdellatif et al. in [117], Lee et al. in [118], and Meivel et al. in [119] highlight the effectiveness of integrating object detection and IoT devices, such as drones, to enhance urban surveillance and security management. For this purpose, the *You Only Look Once (YOLO)* and *Faster Region-based CNN (Faster R-CNN)* algorithms are applied.

The concept of *Industry 4.0* brings manufacturing processes to be monitored and controlled virtually. Recent research in this area has focused on the detection of intelligent small objects to build a digital twin environment. As an example, Yao et al. in [120] proposed a small object detection model in a manufacturing workshop use case using *YOLOX*.

In the context of smart laboratories, Ali et al. in [121] introduced the implementation of object detection. The proposed system contributes to efficient equipment monitoring and helps ensure compliance with safety protocols.

With the growth of communication technology, object detection can be seamlessly performed on cloud servers in real-time scenarios. Baretto et al. in [122] demonstrated an application for person detections with CCTV cameras on cloud servers using *WebRTC* technology [123]. As the technology continues to evolve, the integration of real-time object detection on cloud servers opens up new possibilities for improved monitoring and decision-making.

Table 5.3 summarizes the characteristics of the object detection techniques used in the papers discussed in this subsection.

YOLO and *Faster R-CNN* are popular algorithms for their computation speed and accuracy in detecting objects in image data. The architecture of *YOLO* processes entire images in a single forward pass through the neural network to enable real-time object detection. On the other hand, *Faster R-CNN* uses a two-stage process, where the first stage proposes regions of interest, and the second stage classifies these regions and refines the bounding boxes to achieve better accuracy. This difference in concepts contributes to the different characteristics of the two algorithms, with *YOLO* being highly efficient for real-time processing, while *Faster R-CNN* focuses on improving accuracy by using a two-stage approach.

Implementing *YOLO* and *Faster R-CNN* typically involves deep learning frameworks such as *TensorFlow* and *PyTorch*. These frameworks are commonly employed within Python environments that seamlessly integrate with *CUDA* for GPU acceleration and supporting libraries such as *Keras* and *OpenCV*. In order to accommodate the high demand for computing resources, a physical server is deployed along with GPUs. This hardware setup ensures more efficient processing and optimization of the algorithms.

According to use cases of IoT applications, object detection processes a captured image to obtain the detected objects in an image file. The detected objects are annotated with bounding boxes, class labels, and confidence scores. Similar to image classification, these techniques require a significant storage capacity for dataset storage. In addition, the data management approach is a critical aspect. The users need to carefully consider whether the results will be stored on temporary or permanent storage mechanisms.

Table 5.3: Key characteristics of object recognition techniques.

Ref.	Algorithms	Software Requirements	Data Types		Processing Methods	Computations
			Input	Output		
[116]	Integration of YOLOv3 and *Multitask CNN (MTCNN)*	Python, TensorFlow, CUDA, and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing	Jetson TX1 with 6.00 GB RAM and NVIDIA Maxwell GPU
[117]	YOLOv5	Python with PyTorch, Apache Kafka, Apache Flink and CUDA	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing, batch processing, and dynamic model deployment	Intel Core i7 CPU with 8.00 GB RAM
[118]	Faster R-CNN	Python with PyTorch and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing	Intel Xeon E5-2680 v3 with 128.00 GB and Nvidia Tesla K40 GPU
[119]	Faster R-CNN and YOLOv3	Python with PyTorch, TensorFlow, CUDA, Keras, and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Image pre-processing	-
[120]	YOLOX	Python with PyTorch, CUDA, and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Image enhancement and feature enhancement	Intel Core i9 CPU with16.00 GB RAM and NVIDIA RTX A4000 GPU
[121]	YOLOv5	Python, TensorFlow, CUDA, and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing	Intel XEON E5-2698 v4 with NVIDIA DGX-1 GPU
[122]	YOLOv3	Python, OpenCV, CUDA, and WebRTC	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing	Intel Core i7 CPU with Nvidia GTX 1050 GPU

5.1.5 Text Spotting

This subsection presents an overview of the papers that focus on application use cases of text-spotting techniques in IoT systems.

5.1.5.1 Introduction

In AI, text spotting refers to the ability to detect and recognize texts within an image [90]. This technique is closely related to object detection, as it includes the recognition and localization of the text regions within images. However, text spotting extends beyond object detection by further extracting the textual content presented in the identified regions. The objective of this technique is to automate the extraction of meaningful information from images containing texts. This is particularly important for real-world applications such as mapping, document analysis, and augmented reality.

Text spotting has a significant role in IoT by enabling the extraction of valuable information in texts from visual data collected by sensors. This capability is particularly valuable for the tasks such as recognizing street signs, license plates, and product labels. By effectively identifying and extracting texts from images, text spotting enhances the intelligence of IoT systems, enabling them to derive meaningful insights and support diverse application use cases. Nevertheless, the implementation of text spotting involves numerous challenges. The wide variety of text appearances, including variations in sizes, lengths, widths, and orientations, poses a significant challenge to the development of effective text-spotting techniques.

5.1.5.2 Use Cases in IoT Applications and Characteristics Overview

The implementation of text spotting in IoT has been thoroughly explored in numerous literature studies. They illustrated the effectiveness of text-spotting algorithms in detecting and recognizing textual information from images for various application use cases.

The seamless integrations of IoT and text-spotting techniques in smart cities play a critical role in improving the efficiency, safety, and functionality of urban environments. The integrations are able to optimize parking management, ensure city safety, and improve public services. Bassam et al. in [124], Wu et al. in [125] utilized *Optical Character Recognition (OCR)* model to extract the textual information about available parking spaces. The works of Glasenapp et al. in [126] and Tham et al. in [127] proposed IoT systems to improve public safety by recognizing license plates from video streams.

Abdullah et al. in [128] and Chang et al. in [129] demonstrated the implementation of the OCR model to assist in recognizing information in medicine labels. In [130], Dilshad et al. applied an OCR model to determine the location of a UAV by analyzing visual data from its surroundings. Meanwhile, Promsuk et al. in [131] implemented a neural network to recognize numbers in seven-segment displays of industrial instruments. Extending this concept, Meng et al. in [132] developed early warning systems for cold chain logistics using text spotting to detect labels on goods. In addition, Cao et al. in [133] demonstrated the application of an OCR model in an infrastructure management scenario. They proposed systems for identifying irregular components on terminal blocks of electrical power equipment cabinets.

Table 5.4 presents the characteristics overview of text-spotting techniques applied in the literature studies discussed in this thesis.

Among the applied text-spotting algorithms, OCR models have proven their effectiveness in extracting textual information from vision-based data. First, these models identify regions within

Table 5.4: Key characteristics of text-spotting techniques.

Ref.	Algorithms	Software Requirements	Data Types		Processing Methods	Computations
			Input	Output		
[124]	OCR model	LabView	Captured images	Recognized text	Image pre-processing, segmentation, and morphology filters	-
[125]	ABCNet OCR	Python, PyTorch, and OpenCV	Captured images	Recognized text	Object detection, anomaly filter module, and real-time data processing	-
[126]	OCR model by OpenALPR API	Python, OpenCV, and OpenALPR API	Captured images	Recognized text	Object Detection, Image pre-processing, feature extraction, segmentation, and real-time data processing	Intel Core i5 CPU with 20.00 GB of RAM and Nvidia GTX 1050 GPU
[127]	Tesseract OCR	Python, OpenCV, CUDA, and TensorFlow	Captured images	Recognized text	Object Detection, Image pre-processing, geofencing, segmentation, and real-time data processing	UP Squared AI Edge X Intel Atom CPU with Intel Movidius Myriad VPU
[128]	EasyOCR with BiLSTM	Python, OpenCV, and TensorFlow	Captured images	Recognized text	Image pre-processing and real-time data processing	AMD Ryzen 5900x CPU with 64.00 GB of RAM and NVIDIA RTX 3080 GPU
[129]	PP-OCR	Python and OpenCV	Captured images	Recognized text	Image pre-processing and parameters optimization	Intel Xeon i5 CPU with 16.00 GB of RAM
[130]	EasyOCR	Python, OpenCV, and PyTorch	Captured images	Recognized text	Object detection, image pre-processing, and real-time data processing	Intel Core i7 CPU with 32.00 GB of RAM and Nvidia RTX 2060 Super GPU
[131]	Neural Network	Python	Captured images	Recognized text	Image pre-processing, feature extraction, and real-time data processing	Intel Core i5 CPU with 8.00 GB of RAM
[132]	OCR model	Python and OpenCV	Captured images	Recognized text	Video pre-processing and real-time data processing	-
[133]	Paddle OCR	Python, PyTorch, and OpenCV	Captured images	Recognized text	Object detection, feature extraction, and segmentation	AMD Ryzen 9 with 32.00 GB of RAM and NVIDIA GeForce RTX 3080

the image where text exists. Then, characters within each identified region are recognized and converted to machine-readable texts. Finally, they output the recognized texts and the image with the bounding boxes indicating the text locations.

Currently, a variety of OCR models are available to address different use cases, such as *TesseractOCR*, *EasyOCR*, *PaddleOCR*, and *PaddlePaddle OCR (PP-OCR)*. These models offer different capabilities and should be selected based on characteristics such as performance, ease of use, and suitability for specific applications. In addition, OCR models can be effectively combined with other algorithms to improve the accuracy. In [128], Abdullah et al. demonstrated the integration of *EasyOCR* models with the *BiLSTM* algorithm to improve text recognition results. These integrations demonstrate the flexibility of OCR models and their ability to integrate with other algorithms to perform specialized use cases.

5.1.6 Auditory Perception

In this subsection, I provide an overview of the integration of auditory perception techniques into IoT through a review of the current literature studies that include application use cases.

5.1.6.1 Introduction

The motivation behind the development of auditory perception in AI is to mimic the human ability to understand and interpret sound. While computer vision enables machines to ”see” by recognizing objects from visual information, auditory perception enables machines to ”hear” and understand auditory information [109]. This capability expands AI applications to perform tasks that involve processing and extracting meaningful information from audio signals. Speech recognition, speaker recognition, sound classification, and environmental sound analysis are integral components of auditory perception. By applying these techniques, AI systems are able to extract and identify the auditory environment. This enables the development of more immersive and interactive applications.

In the IoT context, auditory perception is essential for extracting valuable information from the audio data gathered by IoT devices. In general, this technology allows the devices to analyze the sounds in their environments to identify certain patterns, events, and irregularities. This feature enhances the cognitive capabilities of IoT devices, where the IoT systems are able to trigger automated responses and actions based on the results of this auditory analysis.

5.1.6.2 Use Cases in IoT Applications and Characteristics Overview

In this section, I review the literature studies that presented how algorithms in auditory perception are applied in various application scenarios, such as smart cities [134–136], smart homes [137], and smart environments [138].

The application of auditory perception techniques in smart cities refers to the implementation of audio analysis algorithms for urban security.

Balia et al. in [134] and Yan et al. in [135] introduced an IoT system to identify potential threats and accidents using audio data collected in the urban environment. In [134], Balia et al. used the *Short-Time Fourier Transform (STFT)* algorithm to extract spectrograms as features of audio data. Following a similar approach, Yan et al. in [135] utilized *Deep Neural Network (DNN)* for classifier and *Mel-Frequency Cepstral Coefficients (MFCCs)* for audio feature extraction. In another use case, Ciaburro et al. in [136] proposed a UAV presence detection system using sound analysis.

Polo et al. in [137] and Chhaglani et al. in [138] presented the application of auditory perception techniques to monitor environmental sounds in homes and buildings.

Table 5.5 summarizes the characteristics of the auditory perception techniques applied in the literature studies discussed in this subsection.

Table 5.5: Key characteristics of auditory perception techniques.

Ref.	**Algorithms**	**Software Requirements**	**Data Types**		**Processing Methods**	**Computations**
			Input	**Output**		
[134]	STFT, CNN, FCNN, and Bi-LSTM	Python, TensorFlow, and Keras	Audio spectrograms	Dangerous event classes	Audio pre-processing, feature extraction, and hyperparameter optimization	32.00 GB RAM with Nvidia GeForce GTX 1060 Max
[135]	MFCCs and DNN	Python, TensorFlow and Keras	Audio spectrograms	Accident event classes	Audio pre-processing and feature extraction	Intel Core i5 CPU with 16.00 GB RAM
[136]	CNN	Python and TensorFlow	Audio spectrograms	UAV state classes (Boolean)	Feature extraction	-
[137]	MFCCs and CNN	Python and Keras	Audio spectrograms	Daily living activities classes	Feature extraction and real-time processing	Raspberry Pi
[138]	XGBoost Regressions	Python and Java	Audio in frequency domain	Predicted Air Flow Rate	Filtering and data transformation	Android Mobile Phone

In the field of auditory perception, neural network-based algorithms are commonly used. In computer vision applications, features extracted from an image are typically used as the input. However, to process audio data, CNN analyzes spectrograms of audio data as features. Spectrograms refer to visual representations of the frequency variations of a sound signal over time. They consist of coefficients that capture the spectral characteristics of an audio signal. To obtain spectrograms of audio data, feature extraction algorithms such as *MFCCs* [139] and *STFT* [140] algorithms are integrated. The effectiveness of this integration was demonstrated in works by Balia et al. in [134] and Polo et al. in [137]. In addition, similar to the CNN algorithm, other approaches such as *DNN*, *Fully Connected Neural Network (FCNN)*, and *Bi-LSTM* algorithms also require spectrograms of audio data as the input. The selection among these algorithms depends on several factors, such as the complexity of the auditory task, the size and nature of the dataset, and the desired level of abstraction for feature extraction.

Currently, auditory perception algorithms are implemented using deep learning frameworks along with Python programming environments. Through integration with supporting libraries such as *Keras*, the process of building, training, and deploying algorithms based on neural networks can be simplified. As a result, algorithms for auditory perception have lower computational requirements compared to computer vision applications. Researchers are potentially using edge computing devices such as the *Raspberry Pi* to implement the algorithms.

As I explore the characteristics of auditory perception, the implementation requires specific pre-processing steps before audio data can be analyzed. The steps include filtering, data transformation, and feature extraction, because algorithms are not able to process raw sensor data directly. They require transformed representations of audio data, such as spectrograms, to perform auditory perception effectively. Therefore, selecting appropriate feature extraction approaches can enhance the performance of IoT applications in auditory perception.

5.1.7 Natural Language Processing

This subsection provides an overview of papers on implementations of *Natural Language Processing (NLP)* techniques in IoT systems considering application use cases.

5.1.7.1 Introduction

In the field of AI, *NLP* refers to the ability of computers to understand and interact with human language [141]. The objective of this technique is to enhance the efficiency of communications between humans and computers. This involves computers not only understanding human language but also recognizing the contextual details involved in human communication. Through this process, computers are able to perform actions and generate responses that are associated with human language and communication patterns. *NLP* techniques are mainly divided into *Natural Language Understanding (NLU)* and *Natural Language Generation (NLG)*. *NLU* is concerned with understanding and recognizing the linguistic aspects of natural language, while *NLG* is concerned with generating clear responses in the form of words or sentences to facilitate efficient communication. *NLP* integrates speech recognition, particularly in specific scenarios such as voice control systems, to extend its functionality and potential.

Currently, *NLP* has attracted widespread attention from researchers due to its capabilities. In the IoT context, *NLP* plays an important role in enabling users to control and interact with IoT systems using human language. The integration of *NLP* into IoT applications enables more instinctive and interactive connections between humans and computers. It facilitates voice control, text data analysis, and intelligent assistants in IoT environments.

5.1.7.2 Use Cases in IoT Applications and Characteristics Overview

The implementations of *NLP* in IoT applications have been thoroughly explored by several researchers. They demonstrated the effectiveness of *NLP* approaches in improving communications and interactions between humans and IoT systems.

In smart home applications, ongoing developments use *NLP* techniques to recognize and understand natural language commands spoken by humans accurately. Ismail et al. in [142], Froiz et al. in [143] and Ali et al. in [144] proposed IoT an IoT system that controls home appliances using voice commands. In [142], Ismail et al. adopted a robust combination of the *SVM* algorithm and *Dynamic Time Warping (DTW)* to accurately interpret commands from users' voice audio. Following this, Froiz et al. in [143] *Wav2vec2* and *Whisper* models for speech recognition and the *Bidirectional Encoder Representations from Transformers (BERT)* model for *NLP* technique. Furthermore, Ali et al. in [144] combined the *Google Speech API*, *NLP* model, and logistic regression to recognize both structured and unstructured voice commands.

In the context of smart buildings, Dweik et al. in [145] presented a significant step forward by introducing a voice control system designed to manage devices in buildings autonomously. These smart home and smart building use cases demonstrate the benefits of *NLP* techniques in diverse domains.

Table 5.6 presents the characteristics overview of the *NLP* techniques applied in the literature studies discussed in this thesis.

Table 5.6: Key characteristics of NLP techniques.

Ref.	**Algorithms**	**Software Requirements**	**Data Types**		**Processing Methods**	**Computations**
			Input	**Output**		
[142]	SVM with a Dynamic Time Warping (DTW) algorithm	Python and Java	User's speech audio	Command recognized (string)	Speech recognition and device control	Raspberry Pi
[143]	Wav2vec2, Whisper, and BERT models	Python and TensorFlow	User's speech audio	Command recognized (string)	Speech recognition, device control, model optimization	Raspberry Pi 4 with 2 GB of RAM
[144]	Google Speech API, NLP model, and Logistic Regression	Python, NLTK, and TensorFlow	User's speech audio	Command recognized (string)	Speech recognition and device control	Intel Core i5 CPU with 16.00 GB of RAM and NVIDIA GeForce 830 GPU
[145]	Google Speech API and NLP model using Bag-of-Words (BoW) approach	Python, NLTK, and TensorFlow	User's speech audio	Command recognized (string)	Speaker verification, speech recognition, and device control.	-

The main purpose of *NLP* is to extract information from transcribed spoken sentences. To achieve this task, researchers often employ *NLP* models, as illustrated in the works of Ali et al. in [144] and Dweik et al. in [145]. Typically, an *NLP* model executes multiple steps, such as sentence segmentation, word tokenization, prediction of parts of speech, lemmatization, identification of stop words, definition of relationships between tokens, and recognition of named entities. Nevertheless, it is necessary to emphasize that different *NLP* models, such as *BERT* and *BoW* models, may contain different procedures. For instance, a *BoW* model involves steps including tokenization, stop word removal, token normalization, and vocabulary generation [146].

According to the use cases of IoT applications, the effective implementation of this technology requires the consideration of several key elements. Firstly, the software requirements should be considered. Currently, Python programming provides *Natural Language Toolkit (NLTK)* libraries to perform algorithms in *NLP* techniques. The *NLTK* libraries work together with *TensorFlow* to produce the desired results.

Based on the characteristics of *NLP* explored in Table 5.6, it shows that the implementation of related processes will significantly contribute to achieving optimal results.

A speech recognition algorithm becomes an important process. For this purpose, researchers can use existing models or third-party platforms such as *Whisper* models and the *Google Speech API*. Then, a device control process is required to allow an IoT application to perform the control systems. Finally, the model optimization is able to enhance the trained models. For computational hardware, Table 5.6 shows that *NLP* models can be deployed and executed on either servers or edge devices such as *Raspberry Pi*.

5.1.8 Collaborative AI

In this subsection, I present an overview of the practical use cases of the collaborative AI approach in IoT systems in existing literature studies.

5.1.8.1 Introduction

The development of collaborative AI in IoT systems is driven by the demand to address challenges associated with traditional cloud-based processing. In the traditional model, AI algorithms are executed on centralized cloud servers. As a result, cloud servers require high computational resources to cover all the AI processes. This raises many issues related to latency, communication, connectivity, and privacy concerns [90]. To address these challenges, collaborative AI aims to distribute computational tasks effectively by using both edge computing and cloud resources.

In the IoT context, collaborative AI extends data processing at the edge to reduce latency and bandwidth consumption. The purpose of this technique is not only to optimize resources but also to enable real-time or near-real-time analysis of IoT data. This is important in applications where rapid decision-making is required. This paradigm emphasizes the balanced and efficient use of local and cloud resources. Local processing involves the execution of lightweight AI algorithms for fast analysis, addressing the need for reduced latency. At the same time, the centralized cloud servers handle heavier data processing tasks, ensuring a comprehensive and robust approach to AI computation. The implementation of this collaborative approach indicates improvements towards an AI application in IoT systems, which is more adaptable and efficient.

5.1.8.2 Use Cases in IoT Applications and Characteristics Overview

Here, I conducted a thorough review of the literature studies that explored applications of collaborative AI in IoT systems. They demonstrated the effectiveness of collaborative AI in improving system performance across various application use cases. By leveraging the processing capabilities of both the edge and the cloud, collaborative AI has become a powerful paradigm for improving the efficiency and effectiveness of IoT systems.

In [147], Song et al. presented the implementation of collaborative AI in a monitoring system using *Unmanned Aerial Vehicles (UAVs)* as edge computing and cloud servers. The UAV used a faster *R-CNN* model to detect insulator strings. Then, the images of the insulator strings were transmitted to the cloud server. Finally, the cloud-based system identified defects in the insulator strings using the *Up-Net* model. This collaborative approach effectively optimizes the resources of the UAV and the cloud, which improves the efficiency of the monitoring system.

In [148], Li et al. introduced an edge/cloud collaborative architecture designed for efficient image recognition in the smart agriculture use case. The system employs a lightweight DNN model for object detection at the edge. If the object is not successfully recognized, the image is then transmitted to a server for further processing using more powerful DNNs.

In addition, this technique has been developed for further distributed AI architectures. In [149], Chen et al. introduced a distributed real-time object detection framework for video surveillance systems. This approach allows edge nodes to perform object detection using a *YOLO* model. Then, images of the detected objects are sent to the server to be used to generate a new model. Once the model is generated, it is sent back to the edge nodes. Following this concept, in [150], Loseto et al. proposed edge intelligence components that allow edge devices to perform data training using local data to generate models for early prediction. Using data collected from multiple edge devices, the cloud performs more advanced data training to generate highly accurate models and sends them to the edge devices. These scenarios illustrate the use of collaborative AI to continuously improve AI capabilities at the edge.

Table 5.7 presents the characteristics overview of the collaborative AI applied in the literature studies discussed in this thesis.

Table 5.7: Key characteristics of collaborative AI techniques.

Ref.	Algorithms	Software Requirements	Data Types		Processing Methods	Computations
			Input	Output		
[147]	Faster RCNN (UAV), Up-Net Model (cloud)	Python, TensorFlow, Keras, OpenCV and Caffe	Captured images (UAV and cloud)	Images with bounding box (UAV and cloud)	Image pre-processing, rotation and segmentation on the cloud, image detection on UAVs, and real-time data processing	PC Server with NVIDIA GeForce RTX 2080 Ti (Server)
[148]	DNN	Python and PyTorch	Captured images	Image classes	Image difficulty prediction and model optimizations	-
[149]	YOLOv3	Python, PyTorch, and OpenCV	Captured images	Images with bounding box, class labels, and confidence scores	Real-time data processing and model update capabilities	NVIDIA Jetson Xavier NX (Edge) and Intel Core i7 CPU with 32.00 GB of RAM and Nvidia RTX 2080 GPU (Cloud)
[147]	Multi-layer perceptron regressor	Python, Apache Kafka, TensorFlow and Keras	Time-series data	Predicted amount of silica	Real-time data processing and model update capabilities	Raspberry Pi 4 Model B with 4.00 GB of RAM (Edge) and Intel Xeon CPU E5-2673 with 32.00 GB of RAM (Cloud)

The effective integration of collaborative AI into IoT applications involves several elements. First, the software requirement plays an important role in designing lightweight algorithms specifically for resource-constrained edge devices. Due to its integration capabilities with popular AI frameworks and supporting libraries, Python is often preferred for this application.

The architecture for integration between the edge and the cloud is also important, as highlighted in the work of Chen et al. in [149] and Loseto et al. in [147]. The proposed distributed AI architecture allows local data training and model updates directly from the cloud server. To achieve this, well-designed communication methods with an emphasis on efficient protocols are required.

5.1.9 Integration of AI in IoT Platforms

This section presents a comparison between the *SEMAR* platform and the current state-of-the-art research on the integration of AI in IoT platforms to highlight the proposed ideas described in this chapter. I identified eight literature studies that have similar approaches to the proposed design. In [151], Bu et al. proposed a platform for integrating AI with *Industrial Internet of Things (IIoT)* technologies to monitor and optimize manufacturing processes. In [152], Seshan et al. demonstrated the integration of the *FIWARE* framework [153] with AI capabilities, specifically for anomaly detection in wastewater monitoring applications. The *FIWARE* framework was chosen for device management and data collection processes. In another study presented in [154], Ramallo-Gonzalez et al. proposed an IoT platform for smart healthcare that leverages the *FIWARE* framework, big data technologies, and AI-based data analysis support. Both studies demonstrate the use of existing open-source frameworks to build an IoT platform service.

In [155], Raj et al. developed a framework called *EdgeMLOps* to deploy AI models directly at the edge. In [156], Li et al. introduced an *Artificial IoT (AIoT)* platform for smart agriculture applications that supports the addition of AI models on the edge device. In another study by Rong et al. in [88], the *Sophon Edge* platform was designed for collaborative computing between the cloud and edge devices. This platform enables the updating of AI models. In [157], Liang et al. developed an AIoT platform that facilitates the implementation of various AI models. Utilizing a micro-service architecture, each AI model runs concurrently. It allows the platform to support the combination of multiple AI models into a single dataflow process. Then, in [158], Stavropoulos et al. introduced the integration of machine learning into the IoT platform to create virtual sensors to replace physical sensors.

Table 5.8 shows the comparison of our proposed idea with the eight literature studies. Several parameters are considered, as follows:

- *IoT applications*: represents the use cases of the IoT applications that are implemented in each work.
- *Device management*: represents the ability to allow users to dynamically manage devices connected to the platform (Yes or No).
- *Model management*: represents the ability to manage multiple AI models, including adding and updating models (Yes or No).
- *Support various AI techniques*: indicates that the platform supports AI-driven capabilities across several techniques (Yes or No).
- *Edge device integration*: refers to the ability to deploy AI models to edge device systems (Yes or No).
- *Data types*: represents the specific types of data that can be handled by the platform.

Table 5.8: State-of-art comparison between existing related studies and proposed solution.

Ref.	IoT Application	Device Management	Model Management	Support Various AI Techniques	Edge Devices Integration	Data Types
[151]	Smart Manufacturing	✓	✗	✓	✗	Common data types
[153]	Smart Environments	✓	✗	✗	✗	Common data types
[154]	Smart Healthcare	✓	✗	✓	✗	Common data types
[155]	Various IoT applications	✓	✓	✗	✓	Common data types
[156]	Smart Agriculture	✓	✓	✗	✓	Common data types and image
[88]	Various IoT applications	✓	✓	✗	✓	Common data types and image
[157]	Various IoT applications	✓	✓	✓	✗	Common data types, image and audio
[158]	Smart Homes and Environments	✓	✗	✗	✗	Common data types
Our Work	Various IoT applications	✓	✓	✓	✓	Common data types, image and audio

Regarding the covered IoT application use cases, the works of Raj et al. in [155], Rong et al. in [88], and Liang et al. in [157] have the potential to be used in various IoT applications. This is similar to the *SEMAR* platform, which has been implemented and integrated into various IoT application use cases.

All of the mentioned works, including *SEMAR* IoT platform, provide device management capabilities for adding and removing connected IoT devices. These works also allow for receiving common data types from sensors, such as integer, float, string, boolean, date, and time. However, since IoT devices have a variety of data types, an IoT platform should be able to handle media file data types, including images and audio. The works by Li et al. in [156] and Rong et al. in [88] have demonstrated the ability to receive image frame data. Furthermore, the work of Liang et al. in [157] extends this capability by providing a system that can process both image and audio data. The proposed design for the *SEMAR* platform addresses these concerns by facilitating the collection of both image and audio data types.

An IoT platform may require the ability to manage a large number of AI models and flexibly deploy different models across its processes. For this purpose, robust AI model management capabilities are used to enable users to handle AI models in order to achieve optimal results. The works of Raj et al. in [155], Li et al. in [156], Rong et al. in [88], and Liang et al. in [157] exemplify these capabilities in managing AI models within the flow of IoT data processes. The AI integration in the *SEMAR* platform follows this approach. I develop the functionalities that enable users to manage the versioning of AI models, store them in data storage, and deploy them in the flow of data processing.

Several works have emphasized the importance of AI techniques in IoT platforms by designing systems that support multiple AI capabilities. The works of Bu et al. in [151], Ramallo-Gonzalez et al. in [154], and Liang et al. in [157] have demonstrated that their proposed platform is able to accommodate different types of AI techniques. By supporting multiple AI techniques, an *AIoT* platform can effectively handle the variety of data generated by IoT devices. In line with this concept, the AI capabilities in the *SEMAR* platform are designed to facilitate the seamless integration of multiple AI techniques. With these capabilities, the *SEMAR* platform is able to process different types of data.

In the field of IoT, the current approach involves the utilization of edge computing to perform data processing close to the device. The work by Raj et al. in [155], Li et al. in [156], and Rong et al. in [88] introduced a platform that facilitates the implementation of AI models on edge devices. This approach involves utilizing AI models that are designed to be lightweight in order to accommodate the limited computational resources available on the edge device. Following this idea, I also provide the capability to allow users to deploy their AI model to the edge device through the cloud environment.

5.2 Design of AI Techniques Integration in *SEMAR*

In this section, I present the design of integrating AI techniques in the *SEMAR* IoT server platform.

5.2.1 System Overview

The section presents the design for the seamless integration of AI techniques into *SEMAR* with considering the key characteristics and analysis of each AI technique described in Section 5.1. Figure 5.1 shows the design overview of integrating AI techniques in *SEMAR*. It consists of *AI Model Management*, *Real-Time AI*, and *Batch AI* services. The *AI Model Management* This feature handles the management and monitoring of AI models. It allows users to upload generated models from other machines to the *SEMAR*. The *Real-Time AI* feature performs data processing using AI in real-time scenarios. The *Batch AI* feature enables users to apply AI models to process existing data in the data storage. Furthermore, I show how to implement AI on edge computing devices by integrating the *Edge Device Framework* with *SEMAR*.

The limitation of the proposed method is that the training of the AI model occurs outside the *SEMAR* platform environment. This implies that the platform cannot directly influence or control the performance of the model. As a result, the model may perform less effectively. To mitigate this limitation, the *AI Model Management* feature allows seamless versioning of AI models for deployments, enabling users to effectively track and manage different AI models. Another limitation concerns the compatibility of AI models supported by the *SEMAR* platform. This approach restricts the platform's support to Python-based models only.

5.2.2 AI Model Management

The implementation of AI requires a systematic approach to defining objectives, collecting data, building models, and deploying them for real-world applications. As an IoT development tool, *SEMAR* should perform the functions that allow users to implement AI techniques in the applications easily. The current implementation of *SEMAR* provides the ability to collect sensor data. To help the integration of AI models, I design the implementation of the *AI Model Management* feature in *SEMAR*. It allows users to manage, add, remove, and deploy models through the user interfaces of *SEMAR*.

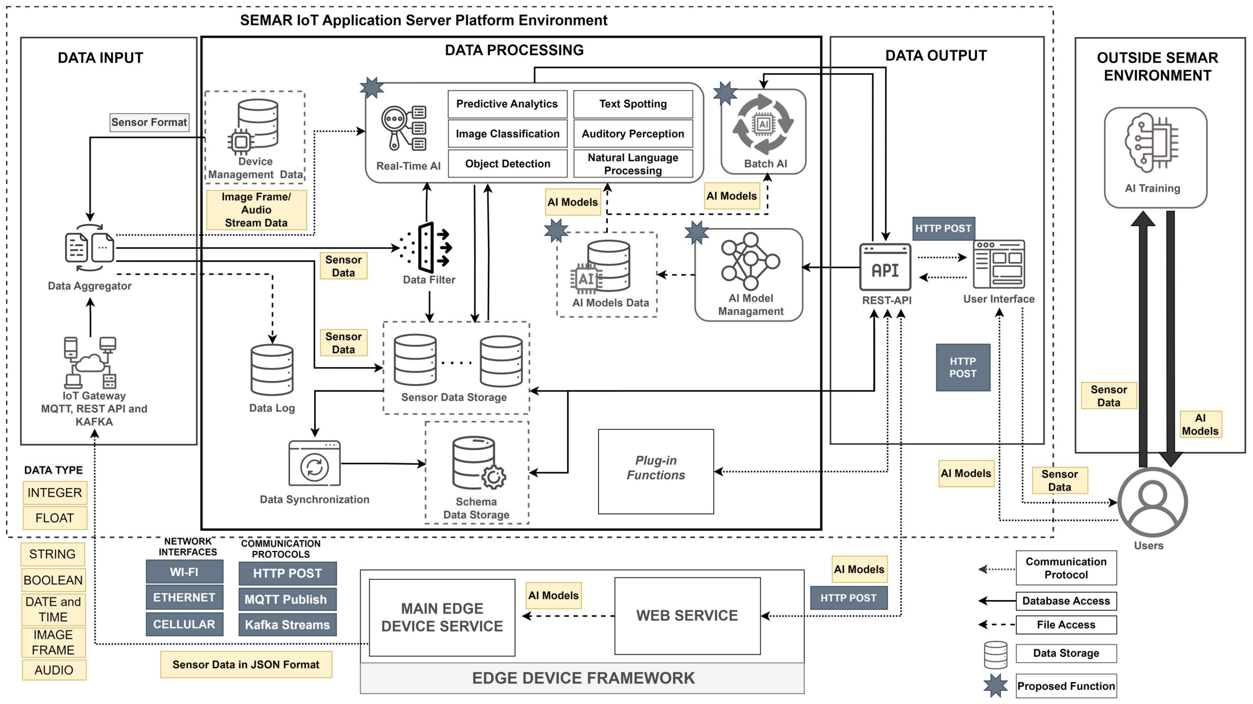


Figure 5.1: Design overview of AI techniques in *SEMAR* IoT application server platform.

In this design, *SEMAR* enables users to utilize models generated on other machines. First, users can grab sensor data stored in the data storage of *SEMAR*. This provides easy access to relevant data sets and simplifies the data preparation process. Then, users engage in the data training process to generate an AI model. After obtaining a trained model, users upload it through user interfaces of *SEMAR*. Next, users define the properties of the model, including its name, version, inputs, outputs, and the type of AI techniques employed. Inputs represent the list of data to be processed, while outputs represent the list of results obtained after AI processing. The system simplifies the deployment process by automatically specifying the data types of inputs and outputs for image classification, object detection, text spotting, and *NLP* techniques, although users are still able to customize these settings. For predictive analytics, users manually define input and output elements. Once a user registers a new model, the system stores it in the AI model data store through file access. This ensures that the model can be easily accessed for future use. Finally, to bring the models into applications, users are allowed to deploy the models for *Real-Time AI* or *Batch AI* processing.

According to the software requirement characteristics identified in the literature review, I select Python to build the features. Then, I build new features in user interfaces and *REST API* services of the *SEMAR* that focus on seamlessly guiding users in managing AI models.

5.2.3 Real-Time and Batch AI Processing

This section introduces two features, namely *Real-Time AI* and *Batch AI* services, for applying AI models in *SEMAR*. The *Real-Time AI* service is specifically designed for scenarios that require immediate AI processing. As shown in Figure 5.1, this service is seamlessly integrated with *IoT cloud gateway* and the *data aggregator* to perform the data stream processing.

In object detection, image classification, text spotting, audio recognition, and *NLP* techniques, the *IoT cloud gateway* receives image frames or audio data. The *data aggregator* verifies the format of the data in following standards such as *JPEG* and *WAV*. The verified data are sent to the *Real-Time AI* service through established communication protocols for further processing by AI models. For this purpose, I utilize the *Kafka* communication protocols. Once the system receives the results, it stores them in the data storage. In the predictive analytics scenario, the data aggregator forwards the data to the data filter before reaching the *Real-Time AI* services, which perform data pre-processing. These services predict the future events of data using an AI model and the historical data collected from the data storage.

The *Batch AI* service provides a service designed for AI processing on existing data saved in the data storage. This service is particularly useful for dealing with large data sets that cannot be processed in real-time scenarios. Unlike the *Real-Time AI* services that process data automatically, users should first select specific data that will be processed through the user interface of *SEMAR*. Then, users select a suitable AI model from the storage. The system systematically applies the AI processes to all of the selected data, collects the results, and saves them back to the data storage. The implementation of *Real-Time AI* and *Batch AI* services in *SEMAR* enhances system flexibility by handling both immediate processing tasks and post-processing analysis requirements.

5.2.4 AI Implementation in Edge Devices

In chapter 4, I introduced the *edge device framework* as one feature of *SEMAR*. This framework allows users to optimize the functionality of IoT devices by allowing them to be configured remotely from the server. It provides the functions for connecting to multiple IoT sensors, processing data in standard formats, and using the collected data through multiple output components. These functions are organized into the *input*, *processing*, and *output* components. As the insights from collaborative AI techniques in the literature review, I design a strategic extension by implementing AI models on edge devices. Figure 5.2 illustrates the design overview of the AI implementation in the edge device framework. I add the *Real-Time AI* function within the processing components to facilitate the immediate processing of sensor data using AI models.

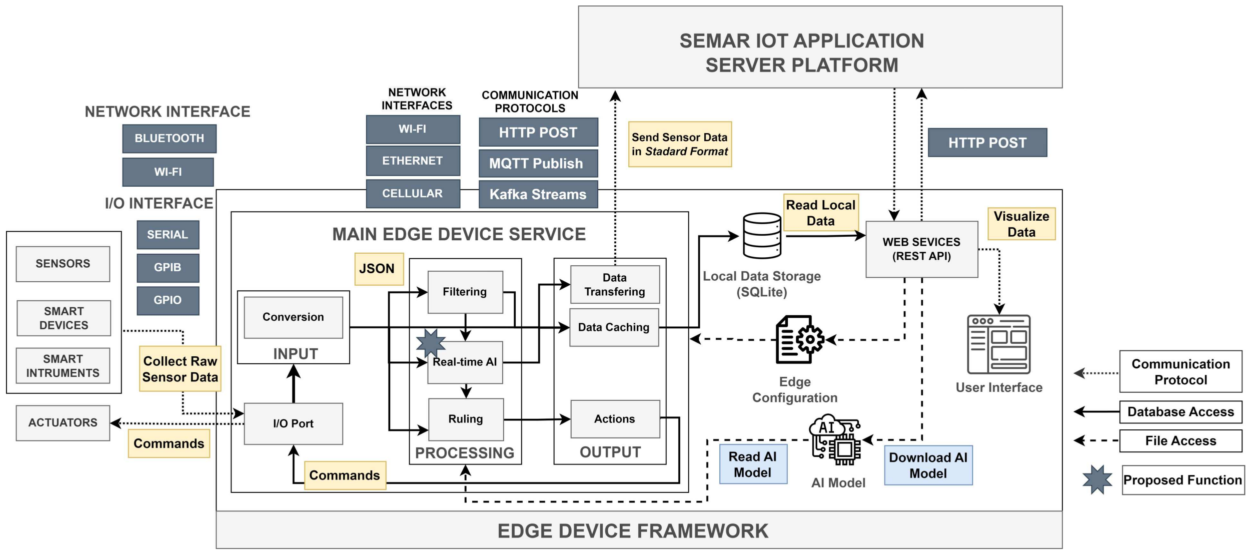


Figure 5.2: Design overview of AI model implementation in edge device framework.

For the installation process, users need to select the appropriate AI model and define field output to store the result. Then, web services download the AI models from the *SEMAR* server through the *HTTP-POST* communication. After the download process is completed, the *main service* on the framework reads the AI model and runs the service to collect sensor data. Finally, once the *Real-Time AI* function receives sensor data from input components or the *filtering* function, it processes the data using this AI model. The generated results are forwarded to the output components or serve as the input to the *ruling* function. This interconnected workflow enables dynamic and responsive integration of AI models at the edge.

5.3 Use Cases of Integration AI and IoT applications in *SEMAR*

In this section, I discuss IoT application use cases that are integrated with the AI-driven capabilities in *SEMAR*. Each use case presents the application overview, requirements, the AI algorithms being employed, and how *SEMAR* can be utilized to support them. They include the drone-based building monitoring system and the *air-conditioning guidance system (AC-Guide)*.

5.3.1 Drone-Based Building Monitoring System

As the first application use case, I discuss the implementation of AI within a drone-based surveillance system. In the building inspection use case, drones emerge as powerful tools for rapid surveillance systems with the ability to navigate autonomously based on missions defined by the programs or to be operated under human control. Their versatility is advantageous for expanding the coverage areas of monitoring systems. By seamlessly integrating drones with AI technologies and an IoT system, drones are able to automatically detect defects in buildings, such as cracks. Figure 5.3 illustrates the system overview of the integration of drone and *SEMAR* for the building crack detection system.

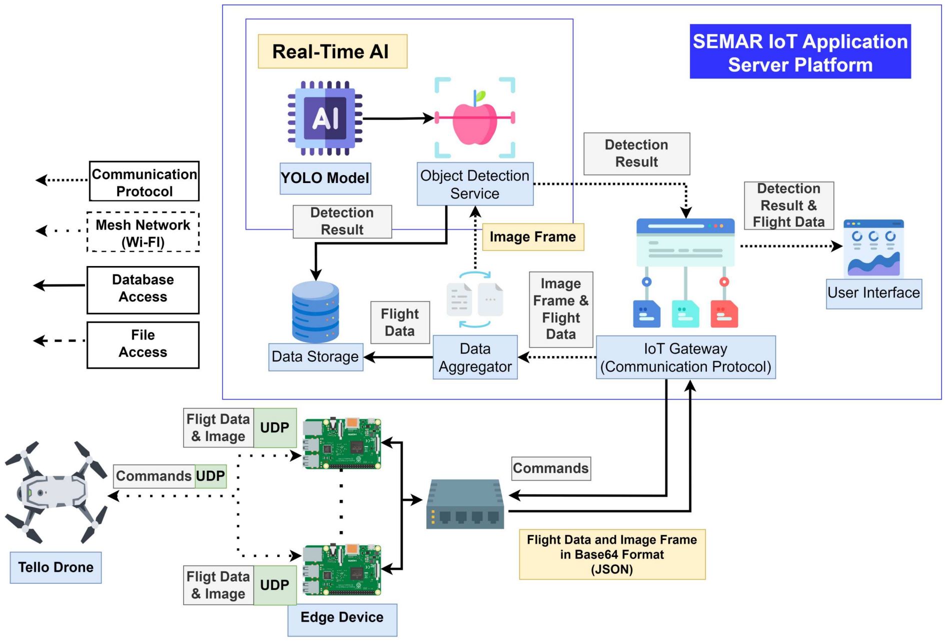


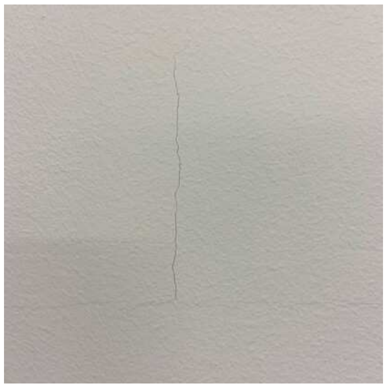
Figure 5.3: System overview of drone-based building monitoring system.

This system consists of flying drones and edge devices connected to the *SEMAR* server. The flying drones capture the image data around the building, while the edge devices are placed in a specific area of the building to control the drones and receive data from it. The drone transmits flight data and images to edge devices through UDP communications over Wi-Fi connections. Edge devices receive and send them to the *SEMAR* server using communication protocol services over ethernet connections. The *IoT Gateway* manages data communications between edge devices and the *SEMAR* server, as well as data distributions in the *SEMAR* server. Once the *SEMAR* server receives the image data, it processes the data using object detection models to detect cracks in the images. Finally, the user interface visualizes the crack detection results.

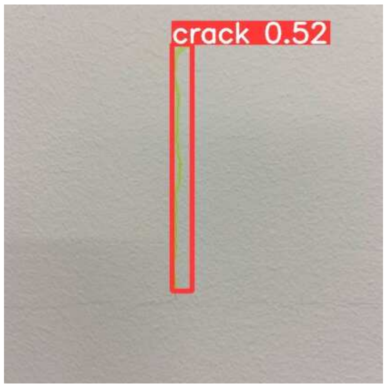
According to Figure 5.3, several services of the *SEMAR* server are required to build this application, such as the communication protocol, object detection, data storage, and user interface services. The communication protocol service should be responsible for transmitting image data. Then, the object detection service provides the capability to identify the cracked objects in the image.

To achieve an efficient implementation of this use case, I will use *Confluent Kafka* as the communication protocol service between *SEMAR* and edge devices. As mentioned in Section 5.1, the *YOLO* algorithm has gained popularity for its efficiency in real-time scenarios. Thus, a *YOLO* model is generated using the open-source crack datasets available on the Internet in [160]. I then upload the *YOLO* model to the *SEMAR* and deploy it to the *Real-Time AI* service to perform crack detections. This deployment step places the crack-detecting capabilities derived from the *YOLO* algorithm into the flow of data streams in *SEMAR*. Once the system detects cracked objects in an image, it stores them in the data storage. Finally, the user interface visualizes the detected cracks for users continuously receiving the results through the communication protocol service.

Figure 5.4 shows the preliminary implementation results of crack detections using the *YOLOv7* algorithm. The crack images were captured using the *Ryze Tello* drone. The model was built using 3717 crack images. Then, I utilized an additional 200 images to validate the model with the *Intersection over Union (IoU)* threshold of 50%. The validation results indicate that the model correctly classified 208 positive samples (true positives), while 41 positive samples were incorrectly classified (false negatives), and 38 negative samples were detected as cracks (false positives). Thus, the model achieved the accuracy up to 73% on average and the *mean average precision (mAP)* up to 82%.



(a) Crack image captured by drone.



(b) Crack detection result.

Figure 5.4: Drone-based crack detection result.

5.3.2 Air-conditioning Guidance System

As the last use case of IoT applications, I introduce the integration of AI in the air conditioning guidance system, namely *AC-Guide*. Recently, the development of smart homes has gained significant popularity and attention from both academia and industry. It focuses on technologies that enhance the convenience, efficiency, and comfort of users' daily lives. In this use case, the environmental condition data are essential for the system to recognize the comfort or discomfort state of the specific area. Therefore, a smart home system utilizes IoT sensors to measure environmental conditions in order to fulfill this important requirement.

Previously, the section 3.10 introduced the integration of *AC-Guide* with the *SEMAR* IoT application platform. The *AC-Guide* system provides a guide to the use of AC by identifying the discomfort state of the room.

The implementation results demonstrate that the system has successfully identified the current state of DI. However, this system can be improved by using advanced analysis techniques to identify data patterns for predicting future events. This improvement is useful in preventing discomfort states.

One AI technique that can be applied to this application is predictive analytics. As mentioned in Section 5.1, predictive analytics allows the system to estimate future values and use the results to support decision-making. Therefore, I propose the application of predictive analytics techniques to perform early guidance systems for the use of AC by considering the prediction data.

For this purpose, I use the *LSTM* algorithm to generate AI models for predicting indoor temperature and humidity using collected sensor data. After the models are generated and uploaded into *SEMAR*, the system deploys them to the *Real-Time* AI service. Once it receives the sensor data, it predicts the future value by using the AI models and the data stored in the data storage. Then, it stores the data in the data storage. Finally, I build the *plug-in* function to calculate the DI state based on the predicted sensor values.

In this thesis, I present the preliminary system to predict the humidity and temperature data collected by the *AC-Guide* system. I trained a model based on the *LSTM* algorithm for 100 epochs. The data sets contain 132 data for training and 68 data for validation. The validation results indicate that the model can predict the values with a *root mean square error (RMSE)* of 0.08 for humidity and 0.04 for temperature. Figure 5.5 shows the results of the predicted values compared to the real value.

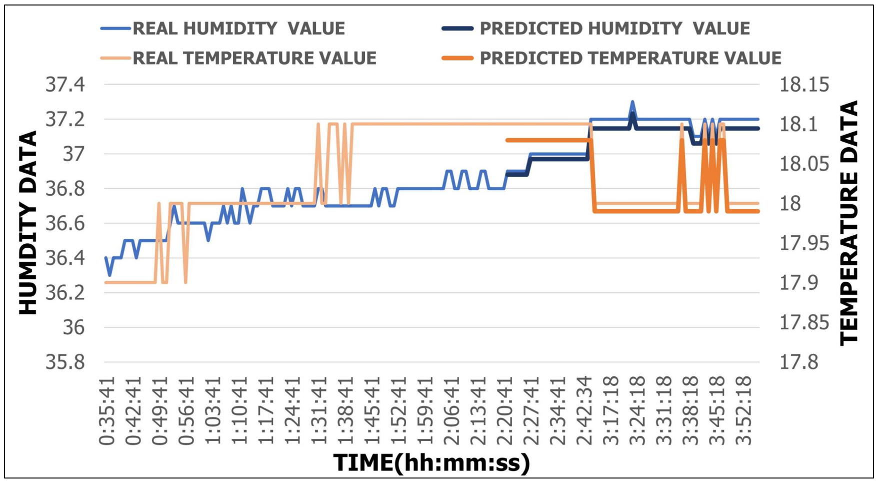


Figure 5.5: Predictive analytics results using LSTM algorithm in AC-Guide system.

5.4 Summary

This chapter presented the study of AI techniques and their implementation use cases for designing AI integrations in the *SEMAR* IoT application server platform. It provided a comprehensive review of current research on the implementation of AI techniques in IoT applications, covering predictive analytics, image classification, object detection, text spotting, auditory perception, *NLP*, and collaborative AI. The characteristics of each technique were identified using key parameters consisting of software requirements, *input/output (I/O)* data types, processing methods, and computations. Based on the findings, the integration of AI techniques into the *SEMAR* and edge device framework was designed. This involves new features, including *AI Model Management*, *Real-Time AI*, and *Batch AI* services. The advantages of the proposed system are described through the integration design with two IoT application use cases.

Chapter 6

Related Works in Literature

This chapter introduces relevant works in literature for this thesis. Several works have discussed the paradigm of IoT architecture, exploring the layered components of the IoT ecosystem and their interactions. A significant amount of researches has been presented to design innovative frameworks for each component of the IoT ecosystem, including cloud and physical layers. Furthermore, recognizing the pivotal roles of AI in the evolution of IoT technology, numerous studies have explored the integration of AI into IoT systems to enhance their functionality and efficiency.

In [162], Kamienski et al. proposed a three-layered Open IoT ecosystem approach for smart application architectures. It includes *input*, *process*, and *output* in IoT application systems. The *input* gathers information from multiple sources, such as sensors and other services. The standard communication protocols cover the device connections. The *process* is given by a collection of methodologies, procedures, and algorithms for effective and efficient data processing. The *output* provides capabilities for data visualizations and accessibility.

In [26], Iera et al. introduced the *Social Internet of Things (SIoT)* architecture paradigm. This architecture comprises IoT applications in objects that are registered on a social networking platform, where each object collaborates and interacts with other objects to provide specialized services. The architecture includes three elements: objects, gateway, and an *SIoT* server. Each component may consist of three layers: sensing, network, and application. It enables IoT objects to conduct high-computational processes, in contrast to only the server performing these tasks. As a common IoT architecture, the network layer is only used to connect the server and the objects. However, this architecture allows the integrations between IoT objects and provides interfaces for IoT objects and humans through network layers. Thus, it provides the developments of IoT applications that interact with one another. This architecture can be considered a reference for improving the design of the IoT application system architecture proposed in this thesis.

In [53], Toma et al. proposed an IoT platform for monitoring air pollution in smart cities. The system contains wireless and wired connections with sensors to send data through *MQTT* communications to the server using cellular networks. It allows sharing data through *REST API*; however, this platform was built and implemented only for this IoT application of monitoring air pollution.

In [64], Senožetnik et al. proposed a management framework for groundwater data in smart cities. The system uses a web-based IoT service to receive data through *HTTP POST*, convert it into the JSON format, and store it in the *MongoDB* database. It also allows sharing collected data through *REST API*. This system is similar to our proposed one; however, the system only provided data communications through *HTTP POST*. Moreover, it did not implement data processing functions to analyze the obtained data.

In [71], Javed et al. proposed an IoT platform for smart buildings. It consists of the discovery, storage, and service planes. The discovery plane performs connectivity controls with devices through *HTTP* communications. The storage plane manages data storage using *Apache Cassandra* [163]. The service plane provides data processing composed of data indexing, visualizing, and analysis.

In [164], Kazmi et al. proposed a platform that provides interoperability of diverse IoT applications in smart cities named *VITAL-OS*. It can be integrated with other IoT application systems through *REST API*.

In [165], Badii et al. proposed an open-source IoT framework architecture for smart cities called *Snap4City*. The system offers modules for device managements, data processing, data analysis, and data visualizations. The authors identified several parameter requirements for developing an IoT platform, including data access, analytics, scalability, and multiple protocols on IoT application systems.

In [166], Mach et al. proposed the concept of the mobile edge computing, which enables IoT applications to perform massive data processing at the device level. However, developers should consider three key aspects, namely, the computational decision, the resource allocation for computational processes, and the mobility management. This approach can reduce the latency of the network in IoT application systems.

In [167], Oueida et al. proposed an integration of the edge computing device and the cloud service in the smart healthcare system. Edge computing was used to gather information from smart devices, process it to obtain the necessary data, and transmit it to the cloud server. The proposed system was suitable for emergency departments and other types of queuing systems.

In [5], Berta et al. proposed a general end-to-end IoT platform that is composed of the cloud-based service for managing sensor data and devices of IoT applications called *Measurify*, and the tool for facilitating the construction of edge devices called *Edgine*. *Edgine* requests the local configuration and executable scripts. Then, it collects data from the sensors, processes them using downloadable scripts and stores it in the cloud. The proposed system has been installed and used in several IoT application systems. The results demonstrated the efficiency of the system by enabling developers to focus on application requirements and design decisions to define the edge system rather than on implementations.

In [168], Kim et al. proposed plug-and-play in IoT platforms, using a web page to manage IoT devices. They utilized *Arduino* boards as edge devices that were connected to the sensors and actuators. The proposed system allows configuring the device for data collection or control actions by accessing the platform website. The implementation results indicate that the system was able to reduce the deployment complexity and increase the robustness of the IoT environment. However, they only considered the device layer and did not address the data visualization and analysis at the cloud level.

In [169], Yang et al. proposed an edge computing framework that is suitable for IoT device developments. This framework provides functions to configure the module hardware security, the data conversion, control, and communication to the server. It also offers advanced data processing capabilities at the edge computing level, including rule engines, data analysis, and application integration. By accessing to the cloud service, this framework allows users to update the configuration through *MQTT* communications. This approach is similar to our method for updating the configuration remotely.

In [90], Zhang et al. introduced the paradigm of *artificial intelligence of things (AIoT)*, which combines AI into an IoT application system. The authors discussed the advantages of implementing AI in IoT, which addresses the challenges in IoT, such as heterogeneous data processing, advanced analysis, and intelligent decision-making. Furthermore, they reviewed AI techniques that were successfully integrated into IoT systems, including computer vision, speech recognition, and *NLP*.

In [92], Abioye et al. provided a comprehensive review of AI applications in the construction industry. This review takes a qualitative approach by examining publication trends for AI and its subfields. It examined AI techniques commonly used in the industry, such as knowledge-based systems, computer vision, robotics, and optimization. Their findings show that implementations of AI can enhance data processing, analyze complex patterns, and provide informed decisions to improve the manufacturing process.

In [94], Alahi et al. provided a comprehensive overview of the integrations of AI and IoT in smart city use cases. This study explored various AI algorithms and their suitability for smart city applications, including NLP, computer vision, machine learning, deep learning, reinforcement learning, and genetic algorithms. By examining key aspects of IoT architecture, applications, network communication, and AI technologies, the authors provide valuable insights into the future potential of integrating IoT and AI for smart cities. Their findings suggest that the synergy between AI and IoT can lead to improvements in the quality of life for urban residents.

Chapter 7

Conclusion

This thesis presented the integrated IoT server platform called *Smart Environmental Monitoring and Analytical in Real-Time (SEMAR)* to support the developments of various IoT application systems using different standards. The *SEMAR* platform provides the standard features for collecting, displaying, processing, and analyzing sensor data from various IoT devices.

Firstly, I presented the design and implementation of the standard functions in *SEMAR* that facilitate the development of the cloud layer for IoT systems. It provides integration functions in *Big Data* environments for data aggregations, synchronizations, and classifications with *machine learning*, as well as *plug-in* functions that access to the data through *REST API*. It utilizes *MQTT* and *REST API* for the communication protocol services. For evaluations, the platform was implemented and integrated with five IoT application systems. Then, the performance and the comparative analysis were conducted. The results show that *SEMAR* provides the higher flexibility and interoperability with the functions for IoT device managements, data communications, decision making, synchronizations, filtering and *plug-in* function capabilities. It confirms the effectiveness and efficiency of the proposed system.

Secondly, I presented the design and implementation of the *Edge Device Framework* in the *SEMAR* IoT application server platform. This framework allows users to remotely optimize the device utilizations by configuring them through the *SEMAR* interface. It works in the *initialization*, *service*, and *update* phases. By utilizing digital signal processing techniques, the filtering functions were implemented in the framework as a data processing component. The framework allows users to define the connectivity of sensor interfaces, filtering functions, sensor element transmissions, communication protocols, local data storage, local visualization, and data transmission intervals on the server. For evaluations, the framework was applied to two IoT application systems. Then, its computational performances were investigated and its features were compared with similar research works. The results show that the proposed framework achieves good performances at all phases. It verified the adaptability and validity of the proposed framework. It provides advanced features and capabilities that are valuable for supporting the developments of the device sensors in IoT application systems.

Finally, I presented a comprehensive review of current researches on AI techniques and their implementation use cases for designing AI integrations in the *SEMAR* IoT application server platform. It covered predictive analytics, image classification, object detection, text spotting, auditory perception, *NLP*, and collaborative AI. I identified the characteristics of each technique using key parameters consisting of software requirements, *input/output (I/O)* data types, processing methods, and computations. Based on the findings, I designed the integrations of AI techniques into the *SEMAR* and the edge device framework. This integration involves new features that include *AI*

Model Management, *Real-Time AI*, and *Batch AI* services. The advantages of the proposed system are described through the integration design with two IoT application use cases, highlighting how *SEMAR* can be used to support them.

In future works, I will continue to develop *SEMAR* by implementing AI technologies to complete the proposed design. Then, I will implement feedback functions with the PID control and the sequence control within the edge computing framework in *SEMAR* to enhance the functionality of the system for *Industry 4.0*.

Bibliography

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): a vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] J. A. Stankovic, "Research directions for the Internet of Things," IEEE Internet of Things Journal, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[3] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in Internet of Things: Taxonomies and open challenges," Mobile Networks and Applications, vol. 24, no. 3, pp. 796–809, Jul. 2018.

[4] J. Cubo, A. Nieto, and E. Pimentel, "A cloud-based Internet of Things platform for Ambient Assisted Living," Sensors, vol. 14, no. 8, pp. 14070–14105, Aug. 2014.

[5] R. Berta, F. Bellotti, A. De Gloria, and L. Lazzaroni, "Assessing versatility of a generic end-to-end platform for IoT ecosystem applications," Sensors, vol. 22, no. 3, p. 713, Jan. 2022.

[6] H. Yar, A. S. Imran, Z. A. Khan, M. Sajjad, and Z. Kastrati, "Towards smart home automation using IoT-enabled edge-computing paradigm," Sensors, vol. 21, no. 14, p. 4932, Jul. 2021.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[8] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," Future Generation Computer Systems, vol. 78, pp. 680–698, Jan. 2018.

[9] S. Salkic, B. C. Ustundag, T. Uzunovic, E. Golubovic, "Edge Computing Framework for Wearable Sensor-Based Human Activity Recognition," IAT 2019, Sarajevo, Bosnia-Herzegovina, 20–23 June 2019; pp. 376–387.

[10] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," IEEE Network, vol. 32, no. 1, pp. 61–65, Jan. 2018.

[11] A. Das, S. Patterson, M. Wittie, "Edgebench: Benchmarking edge computing platforms," In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018.

[12] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, E. de la Torre, "FPGA-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The ARTICO3 framework," Sensors, vol. 18, no. 6, p. 1877, Jun. 2018.

[13] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, “Neuroscience-inspired Artificial Intelligence,” Neuron, vol. 95, no. 2, pp. 245–258, Jul. 2017.

[14] Y. Duan, J. S. Edwards, and Y. K. Dwivedi, “Artificial Intelligence for decision making in the era of big data – evolution, challenges and research agenda,” International Journal of Information Management, vol. 48, pp. 63–71, Oct. 2019.

[15] M. R. Belgaum, Z. Alansari, S. Musa, M. Mansoor Alam, and M. S. Mazliham, “Role of artificial intelligence in cloud computing, IoT and SDN: Reliability and scalability issues,” International Journal of Electrical and Computer Engineering (IJECE), vol. 11, no. 5, p. 4458, Oct. 2021.

[16] T. J. Saleem and M. A. Chishti, “Deep learning for the Internet of Things: Potential benefits and use-cases,” Digital Communications and Networks, vol. 7, no. 4, pp. 526–542, Nov. 2021.

[17] Y. Y. F. Panduman, N. Funabiki, P. Puspitaningayu, M. Kuribayashi, S. Sukaridhoto, W. C. Kao, “Design and implementation of SEMAR IoT server platform with applications,” Sensors, vol. 22, no. 17, p. 6436, Aug. 2022.

[18] Y. Y. Panduman, N. Funabiki, P. Puspitaningayu, M. Sakagami, and S. Sukaridhoto, “Implementations of integration functions in IoT Application Server Platform,” Fifth ICVEE, Sep. 2022.

[19] MQTT Org. “Message Queuing Telemetry Transport Protocol”. `http://mqtt.org/` [Accessed: May 14, 2024].

[20] Y.Y.F. Panduman, N. Funabiki*, S. Ito, R. Husna, M. Kuribayashi, M. Okayasu, J. Shimazu, S. Sukaridhoto, “An edge device framework in SEMAR IoT Application Server Platform,” Information, vol. 14, no. 6, p. 312, May 2023.

[21] W. C. Kao, “An Overview of Edge Device Framework in SEMAR IoT Application Server Platform,” Proc. IEICE General Conf., Saitama, Japan, 2023.

[22] Y. Y. Panduman, N. Funabiki, and S. Sukaridhoto, “Implementation of digital filter functions in edge device framework for IoT application system,” 2023 Sixth ICVEE, Oct. 2023.

[23] Y. Y. Panduman, N. Funabiki, E. D. Fajrianti, S. Fang, and S. Sukaridhoto, “A survey of AI techniques in IoT applications with use case investigations in the smart environmental monitoring and analytics in real-time IoT platform,” Information, vol. 15, no. 3, p. 153, Mar. 2024.

[24] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A survey on enabling technologies, protocols, and applications,” IEEE Communications Surveys Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.

[25] M. Lombardi, F. Pascale, and D. Santaniello, “Internet of Things: A general overview between architectures, protocols and applications,” Information, vol. 12, no. 2, p. 87, Feb. 2021.

[26] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The Social Internet of Things (SIoT) – when social networks meet the internet of things: Concept, architecture and network characterization,” Computer Networks, vol. 56, no. 16, pp. 3594–3608, Nov. 2012.

[27] F. Cauteruccio, L. Cinelli, G. Fortino, C. Savaglio, G. Terracina, D. Ursino, L. Virgili, "An approach to compute the scope of a social object in a Multi-IoT scenario," Pervasive and Mobile Computing, vol. 67, p. 101223, Sep. 2020.

[28] G. S. Chalapathi, V. Chamola, A. Vaish, and R. Buyya, "Industrial Internet of Things (IIoT) applications of edge and fog computing: A review and future directions," Fog/Edge Computing For Security, Privacy, and Applications, pp. 293–325, 2021.

[29] H. Wu, C. Chen, and K. Weng, "Two designs of automatic embedded system energy consumption measuring platforms using GPIO," Applied Sciences, vol. 10, no. 14, p. 4866, Jul. 2020.

[30] V. Mannoni, V. Berg, and F. Dehmas, "A flexible physical layer for LPWA applications: Simulations and field trials," 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Apr. 2019.

[31] F. Muteba, K. Djouani, and T. Olwal, "A comparative survey study on LPWA IoT technologies: Design, considerations, challenges and solutions," Procedia Computer Science, vol. 155, pp. 636–641, 2019.

[32] A. Munshi, "Improved MQTT secure transmission flags in Smart Homes," Sensors, vol. 22, no. 6, p. 2174, Mar. 2022.

[33] D. Dinculeană and X. Cheng, "Vulnerabilities and limitations of MQTT protocol used between IoT devices," Applied Sciences, vol. 9, no. 5, p. 848, Feb. 2019.

[34] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in DevOps," Journal of Systems and Software, vol. 170, p. 110798, Dec. 2020.

[35] Y. Y. F. Panduman, M. R. Ulil Albaab, A. R. Anom Besari, S. Sukaridhoto, and A. Tjahjono, "Implementation of microservice architectures on SEMAR extension for Air Quality Monitoring," In Proceedings of the 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC) 2018, Bali, Indonesia, 29-30 October 2018, pp. 218–224.

[36] A. V. Astafiev, A. L. Zhiznyakov, and A. A. Demidov, "The use of Butterworth filter to compensate for noise in signals from bluetooth low energy beacons in autonomous navigation systems," 2020 International Russian Automation Conference (RusAutoCon), 2020.

[37] M. A. Mahmud, A. Abdelgawad, K. Yelamarthi, and Y. A. Ismai, "Signal processing techniques for IoT-based structural health monitoring," 2017 29th International Conference on Microelectronics (ICM), 2017.

[38] H. Wang, Z. Deng, B. Feng, H. Ma, and Y. Xia, "An adaptive Kalman filter estimating process noise covariance," Neurocomputing, vol. 223, pp. 12–17, Feb. 2017.

[39] F. Wei, Z. Zhang, and K. Jia, "Research on Kalman filter for one-dimensional discrete data," Journal of Physics: Conference Series, vol. 2005, no. 1, p. 012005, Jul. 2021.

[40] P. Kumar, G. P. Gupta, and R. Tripathi, "TP2SF: A trustworthy privacy-preserving secured framework for sustainable smart cities by leveraging blockchain and machine learning," Journal of Systems Architecture, vol. 115, p. 101954, May 2021.

[41] P. Kumar, G. P. Gupta, and R. Tripathi, “An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for IoMT networks,” Computer Communications, vol. 166, pp. 110–124, Jan. 2021.

[42] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2, no. 3, pp. 1–27, Apr. 2011.

[43] A. Suárez Sánchez, P. J. García Nieto, P. Riesgo Fernández, J. J. del Coz Díaz, and F. J. Iglesias-Rodríguez, “Application of an SVM-based regression model to the air quality study at local scale in the Avilés Urban Area (Spain),” Mathematical and Computer Modelling, vol. 54, no. 5–6, pp. 1453–1466, Sep. 2011.

[44] M. M. Ghiasi and S. Zendehboudi, “Decision tree-based methodology to select a proper approach for wart treatment,” Computers in Biology and Medicine, vol. 108, pp. 400–409, May 2019.

[45] . D. Hagan, G. Isaacman-VanWertz, J. Franklin, L. Wallace, B. Kocar, C. Heald, J. Kroll, “Calibration and assessment of Electrochemical Air Quality Sensors by co-location with regulatory-grade instruments,” Atmospheric Measurement Techniques, vol. 11, no. 1, pp. 315–328, Jan. 2018.

[46] W. Wei, O. Ramalho, L. Malingre, S. Sivanantham, J. Little, Mandin, C. “Machine learning and statistical models for predicting indoor air quality,” Indoor Air, vol. 29, no. 5, pp. 704–726, Jul. 2019.

[47] S. Ghosh, A. Dasgupta, A. Swetapadma, “A Study on Support Vector Machine Based Linear and Non-Linear Pattern Classification”. In Proceedings of International Conference on Intelligent Sustainable Systems (ICISS) 2019, Palladam, India, 21–22 February 2019, pp. 24–28.

[48] Eclipse mosquitto, “Eclipse Mosquitto.” https://mosquitto.org/ [Accessed: May 30, 2024].

[49] Dory, M.; Parrish, A.; Berg, B. “Introduction to Tornado.” Sebastopol, USA: O’Reilly Media, 2012.

[50] MongoDB, “Mongodb: The Application Data Platform”. https://www.mongodb.com/ [Accessed: May 30, 2024].

[51] J. Hao and T. K. Ho, “Machine learning made easy: A review of *scikit-learn* package in python programming language,” Journal of Educational and Behavioral Statistics, vol. 44, no. 3, pp. 348–361, Feb. 2019.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software.” Boston, USA: Addison-Wesley, 1994.

[53] Toma, Alexandru, Popa, and Zamfiroiu, “IoT solution for smart cities’ pollution monitoring and the security challenges,” Sensors, vol. 19, no. 15, p. 3401, Aug. 2019.

[54] A. F. Villán, “Mastering OpenCV 4 with Python: A Practical Guide Covering Topics from Image Processing, Augmented Reality to Deep Learning with Opencv 4 and Python 3.7.” Birmingham, UK: Packt Publishing, 2019.

[55] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with TensorFlow: A Review," Journal of Educational and Behavioral Statistics, vol. 45, no. 2, pp. 227–248, Sep. 2019.

[56] S. Huda, N. Funabiki, M. Kuribayashi, R. W. Sudibyo, N. Ishihara, and W.-C. Kao, "A proposal of air-conditioning guidance system using discomfort index," in Proc. Int. Conf. Broad. Wirel. Comput., Commun. Appli., pp. 154-165, 2020.

[57] OpenWeatherMap, "Current weather and forecast - OpenWeatherMap," https://openweathermap.org/ [Accessed on 3 Jun. 2024.].

[58] Y. Huo, P. Puspitaningayu, N. Funabiki, K. Hamazaki, M. Kuribayashi, K. Kojima, "A proposal of the fingerprint optimization method for the fingerprint-based indoor localization system with IEEE 802.15.4 devices," Information, vol. 13, no. 5, p. 211, Apr. 2022.

[59] P. Puspitaningayu, N. Funabiki, Y. Huo, K. Hamazaki, M. Kuribayashi, and W.-C. Kao, "Investigations of detection accuracy improvements for fingerprint-based indoor localization System using IEEE 802.15.4," 4th ICVEE 2021, pp. 1-5, 2021.

[60] Mono wireless, "Mono wireless product information," `https://mono-wireless.com/jp/products/index.html`. [Accessed: Jun. 3, 2024.]

[61] M. Benammar, A. Abdaoui, S. Ahmad, F. Touati, and A. Kadri, "A modular IoT platform for real-time indoor air quality monitoring," Sensors, vol. 18, no. 2, p. 581, Feb. 2018.

[62] D. Hernández-Rojas, T. Fernández-Caramés, P. Fraga-Lamas, and C. Escudero, "A plug-and-play human-centered virtual TEDS architecture for the web of things," Sensors, vol. 18, no. 7, p. 2052, Jun. 2018.

[63] T. Mandava, S.Chen, O. Isafiade, A. Bagula, "An IoT Middleware for Air Pollution Monitoring in Smart Cities: A Situation Recognition Model" In Proceedings of the IST Africa 2018 Conference, Gabarone, Botswana, 9-11 May 2018.

[64] M. Senožetnik, Z. Herga, T.Šubic, L. Bradeško, K. Kenda, K. Klemen, P. Pergar, D. Mladenić, "IoT middleware for water management," EWaS3 2018, Jul. 2018. Proceedings, vol. 2, no. 11, Jul. 2018.

[65] J. Soininen, M.Taumberger, R. Dantas, A.Toscano, T. Salmon Cinotti, R. Filev Maia, A. Torre Neto, "Smart water management platform: IoT-based precision irrigation for agriculture," Sensors, vol. 19, no. 2, p. 276, Jan. 2019.

[66] G. Chiesa, S. Cesari, M. Garcia, M. Issa, and S. Li, "Multisensor IoT platform for optimising IAQ levels in buildings through a smart ventilation system," Sustainability, vol. 11, no. 20, p. 5777, Oct. 2019.

[67] A. de M. Del Esposte, E. Santana, L. Kanashiro, F. Costa, K. Braghetto, N. Lago, F. Kon, "Design and evaluation of a scalable smart city software platform with large-scale simulations," Future Generation Computer Systems, vol. 93, pp. 427–441, Apr. 2019.

[68] I. T. Christou, N. Kefalakis, A. Zalonis, J. Soldatos, and R. Bröchler, "End-to-end industrial IoT platform for actionable predictive maintenance," IFAC-PapersOnLine, vol. 53, no. 3, pp. 173–178, 2020.

[69] I. Marcu, G. Suciu, C. Bălăceanu, A. Vulpe, and A.-M. Drăgulinescu, “Arrowhead technology for digitalization and automation solution: Smart cities and smart agriculture,” Sensors, vol. 20, no. 5, p. 1464, Mar. 2020.

[70] S. Trilles, A. González-Pérez, and J. Huerta, “An IoT platform based on microservices and serverless paradigms for smart farming purposes,” Sensors, vol. 20, no. 8, p. 2418, Apr. 2020.

[71] A. Javed, A. Malhi, T. Kinnunen, and K. Framling, “Scalable IoT platform for heterogeneous devices in smart environments,” IEEE Access, vol. 8, pp. 211973–211985, 2020.

[72] A. Boursianis, M. Papadopoulou, A. Gotsis, S. Wan, P. Sarigiannidis, S. Nikolaidis, S. Goudos, “Smart irrigation system for precision agriculture—the AREThOU5A IoT platform,” IEEE Sensors Journal, vol. 21, no. 16, pp. 17539–17547, Aug. 2021.

[73] M. Antunes, A. Santiago, S. Manso, D. Regateiro, J. Barraca, D. Gomes, R. Aguiar, “Building an IoT platform based on service containerisation,” Sensors, vol. 21, no. 19, p. 6688, Oct. 2021.

[74] A. Depari, D. Fernandes Carvalho, P. Bellagente, P. Ferrari, E. Sisinni, A. Flammini, A. Padovani, “An IoT based architecture for enhancing the effectiveness of prototype medical instruments applied to neurodegenerative disease diagnosis,” Sensors, vol. 19, no. 7, p. 1564, Mar. 2019.

[75] M. T. Thompson, “Analog Low-pass filters,” Intuitive Analog Circuit Design, pp. 531–583, 2014.

[76] P. Podder, Md. Mehedi Hasan, Md. Rafiqul Islam, and M. Sayeed, “Design and implementation of Butterworth, Chebyshev-I and Elliptic filter for speech signal analysis,” International Journal of Computer Applications, vol. 98, no. 7, pp. 12–18, Jul. 2014.

[77] M. B. Prakash, S. V, G. E. A, and S. K. P, “Noise reduction of ECG using Chebyshev filter and classification using machine learning algorithms,” 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2021.

[78] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T.Reddy, et al., “SciPy 1.0: Fundamental algorithms for scientific computing in python,” Nature Methods, vol. 17, no. 3, pp. 261–272, Feb. 2020.

[79] M. Karaim, A. Noureldin, and T. B. Karamat, “Low-cost IMU data denoising using Savitzky-Golay filters,” 2019 International Conference on Communications, Signal Processing, and their Applications (ICCSPA), 2019.

[80] F. Samann and T. Schanze, “An efficient ECG denoising method using discrete wavelet with Savitzky-Golay filter,” Current Directions in Biomedical Engineering, vol. 5, no. 1, pp. 385–387, Sept. 2019.

[81] R. Schafer, “What is a Savitzky-Golay filter? [Lecture notes],” IEEE Signal Processing Magazine, vol. 28, no. 4, pp. 111–117, Jul. 2011.

[82] Graphtec, “Wireless LAN—Midi Logger GL240: Graphtec,” `https://www.graphtec.co.jp/en/instruments/gl240/wireless.html` [Accessed: Jun. 7, 2024.]

[83] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, “SpanEdge: Towards unifying stream processing over central and nearthe-edge data centers,” in Proc. IEEE/ACM Symp. Edge Comput. (SEC), Oct. 2016, pp. 168–178.

[84] S. Banerjee, P. Liu, A. Patro, and D. Willis, “ParaDrop: An Edge Computing Platform in Home Gateways,” Fog for 5G and IoT, pp. 11–23, Mar. 2017.

[85] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, Q. Zhang, “Edge computing in IoT-based manufacturing,” IEEE Communications Magazine, vol. 56, no. 9, pp. 103–109, Sep. 2018.

[86] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta, “Smart farming IoT platform based on edge and cloud computing,” Biosystems Engineering, vol. 177, pp. 4–17, Jan. 2019.

[87] R. Ullah, M. A. Rehman, and B.-S. Kim, “Design and implementation of an open source framework and prototype for named Data Networking-based Edge Cloud Computing System,” IEEE Access, vol. 7, pp. 57741–57759, 2019.

[88] G. Rong, Y. Xu, X. Tong, and H. Fan, “An edge-cloud collaborative computing platform for building AIoT applications efficiently,” Journal of Cloud Computing, vol. 10, no. 1, Jul. 2021.

[89] Z. Sharif, L. T. Jung, M. Ayaz, M. Yahya, and D. Khan, “Smart Home Automation by internet-of-things edge computing platform,” International Journal of Advanced Computer Science and Applications, vol. 13, no. 4, 2022.

[90] J. Zhang and D. Tao, “Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things,” IEEE Internet of Things Journal, vol. 8, no. 10, pp. 7789–7817, May 2021.

[91] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, “A systematic literature review on hardware implementation of Artificial Intelligence Algorithms,” The Journal of Supercomputing, vol. 77, no. 2, pp. 1897–1938, May 2020.

[92] S. O. Abioye, L. Akanbi, A. Ajayi, J.M. Davila Delgado, M. Bilal, O.O. Akinade, A. Ahmed, “Artificial Intelligence in the construction industry: A review of present status, opportunities and future challenges,” Journal of Building Engineering, vol. 44, p. 103299, Dec. 2021.

[93] I. H. Sarker, “Machine learning: Algorithms, real-world applications and research directions,” SN Computer Science, vol. 2, no. 3, Mar. 2021.

[94] M. E. Alahi, A.Sukkuea, F.W. Tina, A. Nag, W. Kurdthongmee, K. Suwannarat, S.C. Mukhopadhyay, “Integration of IOT-enabled technologies and Artificial Intelligence (AI) for Smart City Scenario: Recent advancements and future trends,” Sensors, vol. 23, no. 11, p. 5206, May 2023.

[95] V. Kumar and M. L., “Predictive analytics: A review of trends and techniques,” International Journal of Computer Applications, vol. 182, no. 1, pp. 31–37, Jul. 2018.

[96] Imran, N. Iqbal, S. Ahmad, and D. H. Kim, “Towards mountain fire safety using fire spread predictive analytics and mountain fire containment in IoT environment,” Sustainability, vol. 13, no. 5, p. 2461, Feb. 2021.

[97] A. Hussain, U. Draz, T. Ali, S. Tariq, M. Irfan, A. Glowacz, J.A. Antonino Daviu, S. Yasin, S.Rahman, "Waste management and prediction of air pollutants using IoT and machine learning approach," Energies, vol. 13, no. 15, p. 3930, Aug. 2020.

[98] X.-B. Jin, W.-T. Gong, J.-L. Kong, Y.-T. Bai, and T.-L. Su, "A variational bayesian deep network with data self-screening layer for massive time-series data forecasting," Entropy, vol. 24, no. 3, p. 335, Feb. 2022.

[99] X. Bampoula, G. Siaterlis, N. Nikolakis, and K. Alexopoulos, "A deep learning model for predictive maintenance in cyber-physical production systems using LSTM autoencoders," Sensors, vol. 21, no. 3, p. 972, Feb. 2021.

[100] Y. K. Teoh, S. S. Gill, and A. K. Parlikad, "IoT and fog-computing-based predictive maintenance model for effective asset management in Industry 4.0 using machine learning," IEEE Internet of Things Journal, vol. 10, no. 3, pp. 2087–2094, Feb. 2023.

[101] M. Shorfuzzaman and M. S. Hossain, "Predictive analytics of energy usage by IoT-based smart home appliances for green urban development," ACM Transactions on Internet Technology, vol. 22, no. 2, pp. 1–26, Nov. 2021.

[102] N. Guo, W. Chen, M. Wang, Z. Tian, and H. Jin, "Appling an improved method based on ARIMA model to predict the short-term electricity consumption transmitted by the internet of things (IoT)," Wireless Communications and Mobile Computing, vol. 2021, pp. 1–11, Apr. 2021.

[103] A. A. Nancy, D. Ravindran, P. M. Raj Vincent, K. Srinivasan, and D. Gutierrez Reina, "IoT-cloud-based smart healthcare monitoring system for heart disease prediction via deep learning," Electronics, vol. 11, no. 15, p. 2292, Jul. 2022.

[104] A. F. Subahi, O.I. Khalaf, Y. Alotaibi, R. Natarajan, N. Mahadev, T. Ramesh, "Modified self-adaptive Bayesian algorithm for smart heart disease prediction in IoT system," Sustainability, vol. 14, no. 21, p. 14208, Oct. 2022.

[105] G. Patrizi, A. Bartolini, L. Ciani, V.Gallo, P. Sommella, M. Carratu, "A virtual soil moisture sensor for smart farming using deep learning," IEEE Transactions on Instrumentation and Measurement, vol. 71, pp. 1–11, 2022.

[106] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," Neural Computation, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[107] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," Neural Computation, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[108] A. L. Schaffer, T. A. Dobbins, and S.-A. Pearson, "Interrupted time series analysis using autoregressive integrated moving average (ARIMA) models: A guide for evaluating large-scale health interventions," BMC Medical Research Methodology, vol. 21, no. 1, Mar. 2021.

[109] Y. Xu et al., "Artificial Intelligence: A powerful paradigm for scientific research," The Innovation, vol. 2, no. 4, p. 100179, Nov. 2021.

[110] S. S. Chouhan, U. P. Singh, and S. Jain, “Automated plant leaf disease detection and classification using fuzzy based function network,” Wireless Personal Communications, vol. 121, no. 3, pp. 1757–1779, Jul. 2021.

[111] H. S. Munawar, F. Ullah, S. Qayyum, and A. Heravi, “Application of deep learning on UAV-based aerial images for flood detection,” Smart Cities, vol. 4, no. 3, pp. 1220–1243, Sep. 2021.

[112] M. Abd Elaziz, A. Mabrouk, A. Dahou, and S. A. Chelloug, “Medical Image Classification utilizing ensemble learning and levy flight-based honey badger algorithm on 6G-enabled internet of things,” Computational Intelligence and Neuroscience, vol. 2022, pp. 1–17, May 2022.

[113] A. Y. Saleh, C. K. Chin, V. Penshie, and H. R. Al-Absi, “Lung Cancer Medical Images classification using hybrid CNN-SVM,” International Journal of Advances in Intelligent Informatics, vol. 7, no. 2, p. 151, Jul. 2021.

[114] S. Iyer, T. Velmurugan, A. H. Gandomi, V. Noor Mohammed, K. Saravanan, S. Nandakumar, “Structural health monitoring of railway tracks using IoT-based multi-robot system,” Neural Computing and Applications, vol. 33, no. 11, pp. 5897–5915, Sep. 2020.

S. Iyer,T. Velmurugan, A. H.Gandomi, V. Noor Mohammed, K. Saravanan, S. Nandakumar, “Structural health monitoring of railway tracks using IoT-based multi-robot system,” Neural Computing and Applications, vol. 33, no. 11, pp. 5897–5915, Sep. 2020.

[115] L. D. Medus, M. Saban, J. V. Francés-Víllora, M. Bataller-Mompeán, and A. Rosado-Muñoz, “Hyperspectral image classification using CNN: Application to Industrial Food Packaging,” Food Control, vol. 125, p. 107962, Jul. 2021.

[116] X. Zhou, X. Xu, W. Liang, Z. Zeng, and Z. Yan, “Deep-learning-enhanced multitarget detection for end–edge–cloud surveillance in smart IoT,” IEEE Internet of Things Journal, vol. 8, no. 16, pp. 12588–12596, Aug. 2021.

[117] T. Abdellatif, M. A. Sedrine, and Y. Gacha, “Dromod: A drone-based multi-scope object detection system,” IEEE Access, vol. 11, pp. 26652–26666, 2023.

[118] J. Lee, J. Wang, D. Crandall, S. Šabanovi´c, and G. Fox, “Real-time, cloudbased object detection for unmanned aerial vehicles,” in Proc. 1st IEEE Int. Conf. Robot. Comput., 2017, pp. 36–43.

[119] S. Meivel, N. Sindhwani, R. Anand, D. Pandey, A.A. Alnuaim, A.S. Altheneyan, M.Y. Jabarulla, M.E. Lelisho, “Mask detection and social distance identification using internet of things and faster R-CNN algorithm,” Computational Intelligence and Neuroscience, vol. 2022, pp. 1–13, Feb. 2022.

[120] R. Yao, P. Qi, D. Hua, X. Zhang, H. Lu, X. Liu, “A foreign object detection method for belt conveyors based on an improved YOLOX model,” Technologies, vol. 11, no. 5, p. 114, Aug. 2023.

[121] L. Ali, F. Alnajjar, M.M. Parambil, M.I. Younes, Z.I. Abdelhalim, H. Aljassmi, “Development of YOLOv5-based real-time smart monitoring system for increasing lab safety awareness in educational institutions,” Sensors, vol. 22, no. 22, p. 8820, Nov. 2022.

[122] A. Baretto, N. Pudussery, V. Subramaniam, and A. Siddiqui, “Real-time WebRTC based mobile surveillance system,” International Journal of Engineering and Management Research, vol. 11, no. 3, Jun. 2021.

[123] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology overview and signaling solution design and implementation,” In Proceedings of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 2015, pp. 1006–1009.

[124] R. Bassam and F. Samann, “Smart parking system based on improved OCR model,” IOP Conference Series: Materials Science and Engineering, vol. 978, no. 1, p. 012007, Nov. 2020.

[125] Z. Wu, X. Chen, J. Wang, X. Wang, Y. Gan, M. Fang, T. Xu, “OCR-RTPS: An OCR-based real-time positioning system for the valet parking,” Applied Intelligence, vol. 53, no. 14, pp. 17920–17934, Jan. 2023.

[126] L. A. Glasenapp, A. F. Hoppe, M. A. Wisintainer, A. Sartori, and S. F. Stefenon, “OCR applied for identification of vehicles with irregular documentation using IoT,” Electronics, vol. 12, no. 5, p. 1083, Feb. 2023.

[127] M.-L. Tham and W. K. Tan, “IoT based license plate recognition system using deep learning and OpenVINO,” In Proceedings of the 2021 4th International Conference on Sensors, Signal and Image Processing, Oct. 2021, pp. 7–14.

[128] R. Abdullah, R. Ahmed, and L. Jamal, “A novel IoT-based medicine consumption system for elders,” SN Computer Science, vol. 3, no. 6, Sep. 2022.

[129] J. Chang, H. Ong, T. Wang, and H.-H. Chen, “A fully automated intelligent medicine dispensary system based on AIoT,” IEEE Internet of Things Journal, vol. 9, no. 23, pp. 23954–23966, Dec. 2022.

[130] N. Dilshad, A. Ullah, J. Kim, and J. Seo, “Locateuav: Unmanned aerial vehicle location estimation via contextual analysis in an IoT environment,” IEEE Internet of Things Journal, vol. 10, no. 5, pp. 4021–4033, Mar. 2023.

[131] N. Promsuk and A. Taparugssanagorn, “Numerical reader system for digital measurement instruments embedded industrial Internet of Things,” Journal of Communications, pp. 132–142, 2021.

[132] J. Meng, “Research on the early warning system of cold chain cargo based on OCR Technology,” World Journal of Engineering and Technology, vol. 10, no. 03, pp. 527–538, 2022.

[133] W. Cao, Z. Chen, X. Deng, C. Wu, and T. Li, “An identification method for irregular components related to terminal blocks in equipment cabinet of Power Substation,” Sensors, vol. 23, no. 18, p. 7739, Sep. 2023.

[134] R. Balia, A. Giuliani, L. Piano, A. Pisu, R. Saia, N. Sansoni, “A comparison of audio-based deep learning methods for detecting anomalous road events,” Procedia Computer Science, vol. 210, pp. 198–203, 2022.

[135] L. Yan and S.-W. Ko, “In-tunnel accident detection system based on the learning of accident sound,” The Open Transportation Journal, vol. 15, no. 1, pp. 81–92, May 2021.

[136] G. Ciaburro and G. Iannace, “Improving smart cities safety using sound events detection based on Deep Neural Network algorithms,” Informatics, vol. 7, no. 3, p. 23, Jul. 2020.

[137] A. Polo-Rodriguez, J. M. Vilchez Chiachio, C. Paggetti, and J. Medina-Quero, “Ambient sound recognition of daily events by means of convolutional neural networks and fuzzy temporal restrictions,” Applied Sciences, vol. 11, no. 15, p. 6978, Jul. 2021.

[138] B. Chhaglani, C. Zakaria, A. Lechowicz, J. Gummeson, and P. Shenoy, “Flowsense: Monitoring Airflow in Building Ventilation Systems Using Audio Sensing,” Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 6, no. 1, pp. 1–26, Mar. 2022.

[139] V. Tiwari, “MFCC and its applications in speaker recognition,” Int. J. Emerg. Technol., vol. 1, no. 1, pp. 19–22, Feb. 2010.

[140] H. H. Giv, “Directional short-time fourier transform,” Journal of Mathematical Analysis and Applications, vol. 399, no. 1, pp. 100–107, Mar. 2013.

[141] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 2, pp. 604–624, Feb. 2021.

[142] A. Ismail, S. Abdlerazek, and I. M. El-Henawy, “Development of smart healthcare system based on speech recognition using support vector machine and dynamic time warping,” Sustainability, vol. 12, no. 6, p. 2403, Mar. 2020.

[143] I. Froiz-Míguez, P. Fraga-Lamas, and T. M. Fernández-CaraméS, “Design, implementation, and practical evaluation of a voice recognition based IoT home automation system for low-resource languages and resource-constrained edge IOT devices: A system for Galician and mobile opportunistic scenarios,” IEEE Access, vol. 11, pp. 63623–63649, 2023.

[144] A. A. Ali, M. Mashhour, A. S. Salama, R. Shoitan, and H. Shaban, “Development of an intelligent personal assistant system based on IoT for people with disabilities,” Sustainability, vol. 15, no. 6, p. 5166, Mar. 2023.

[145] W. Dweik, M. Abdalla, Y. AlHroob, A. AlMajali, S.A. Mustafa, M. Abdel-Majeed, “Skeleton of implementing voice control for building automation systems,” Scientific Programming, vol. 2022, pp. 1–15, Sep. 2022.

[146] K. Juluru, H.-H. Shih, K. N. Keshava Murthy, and P. Elnajjar, “Bag-of-words technique in natural language processing: A primer for radiologists,” RadioGraphics, vol. 41, no. 5, pp. 1420–1426, Sep. 2021.

[147] C. Song, W. Xu, G. Han, P. Zeng, Z. Wang, S. Yu, “A cloud edge collaborative intelligence method of insulator string defect detection for power IIoT,” IEEE Internet of Things Journal, vol. 8, no. 9, pp. 7510–7520, May 2021.

[148] M. Li, Y. Li, Y. Tian, L. Jiang, and Q. Xu, "AppealNet: An efficient and highly-accurate edge/cloud collaborative architecture for DNN Inference," In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), Dec. 2021, pp. 409-414.

[149] Y.-Y. Chen, Y.-H. Lin, Y.-C. Hu, C.-H. Hsia, Y.-A. Lian, S.-Y. Jhong, "Distributed real-time object detection based on edge-cloud collaboration for smart video surveillance applications," IEEE Access, vol. 10, pp. 93745–93759, 2022.

[150] G. Loseto, F. Scioscia, M. Ruta, F. Gramegna, S. Ieva, C. Fasciano, I. Bilenchi, D. Loconte, "Osmotic cloud-edge intelligence for IoT-based cyber-physical systems," Sensors, vol. 22, no. 6, p. 2166, Mar. 2022.

[151] L. Bu, Y. Zhang, H. Liu, X. Yuan, J. Guo, S. Han, "An iiot-driven and AI-enabled framework for smart manufacturing system based on three-terminal collaborative platform," Advanced Engineering Informatics, vol. 50, p. 101370, Oct. 2021.

[152] S. Seshan, D. Vries, M. van Duren, A. van Helm, and J. Poinapen, "AI-based validation of wastewater treatment plant sensor data using an open data exchange architecture," IOP Conference Series: Earth and Environmental Science, vol. 1136, no. 1, p. 012055, Jan. 2023.

[153] F. Cirillo, G. Solmaz, E.L. Berz, M. Bauer, B. Cheng, E. Kovacs,"A standard-based open source IoT platform: FIWARE," IEEE Internet of Things Magazine, vol. 2, no. 3, pp. 12–18, Sep. 2019.

[154] A. P. Ramallo-González, A. González-Vidal, and A. F. Skarmeta, "Ciotvid: Towards an open IoT-platform for infective pandemic diseases such as covid-19," Sensors, vol. 21, no. 2, p. 484, Jan. 2021.

[155] E. Raj, D. Buffoni, M. Westerlund, and K. Ahola, "Edge MLOps: An automation framework for AIoT applications," 2021 IEEE International Conference on Cloud Engineering (IC2E), Oct. 2021.

[156] H. Li, S. Li, J. Yu, Y. Han, and A. Dong, "AIoT Platform Design Based on Front and Rear End Separation Architecture for Smart Agricultural", 2022 4th Asia Pacific Information Technology Conference (APIT 2022), pp. 208-214, Jan. 2022.

[157] Y.-C. Liang, K.-R. Wu, K.-L. Tong, Y. Ren, and Y.-C. Tseng, "An exchange-based AIoT platform for fast AI application development," Proceedings of the 19th ACM International Symposium on QoS and Security for Wireless and Mobile Networks, pp. 105-114, Oct. 2023.

[158] [1] G. Stavropoulos, J. Violos, S. Tsanakas, and A. Leivadeas, "Enabling artificial intelligent virtual sensors in an IoT environment," Sensors, vol. 23, no. 3, p. 1328, Jan. 2023.

[159] Y.Y.F. Panduman, N. Funabiki, S. Sukaridhoto, "An Idea of Drone-Based Building Crack Detection System in SEMAR IoT Server Platform," In Proceedings of 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE) 2023, Oct. 2023.

[160] University, "Crack Instance Segmentation Dataset by University," Roboflow, `https://universe.roboflow.com/university-bswxt/crack-bphdr/dataset/2` [Accessed: Jun. 21, 2024]

[161] E. D. Fajrianti, N. Funabiki, S. Sukaridhoto, Y.Y.F. Panduman, K. Dezheng, F. Shihao, A.A.P. Surya, "INSUS: Indoor navigation system using unity and smartphone for user ambulation assistance," Information, vol. 14, no. 7, p. 359, Jun. 2023.

[162] C. Kamienski, R. Prati, J. Kleinschmidt and J.P. Soininen, "Designing an Open IoT Ecosystem", Proceedings of the Workshop on Cloud Networks 2019, Jul. 2019.

[163] Apache Cassandra, "Open source nosql database," Apache Cassandra, `https://cassandra.apache.org/` [Accessed: Jun. 27, 2024]

[164] A. Kazmi, M. Serrano, and J. Soldatos, "Vital-OS: An open source IoT operating system for smart cities," IEEE Communications Standards Magazine, vol. 2, no. 2, pp. 71–77, Jun. 2018.

[165] C. Badii, P. Bellini, A. Difino, and P. Nesi, "Smart city IoT platform respecting GDPR privacy and security aspects," IEEE Access, vol. 8, pp. 23601–23623, 2020.

[166] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," IEEE Commun. Surv. Tutorials, vol. 19, no. 3, pp. 1628–1656, 2017.

[167] S. Oueida, Y. Kotb, M. Aloqaily, Y. Jararweh, and T. Baker, "An edge computing based smart healthcare framework for resource management," Sensors, vol. 18, no. 12, p. 4307, Dec. 2018.

[168] W. Kim, H. Ko, H. Yun, J. Sung, S. Kim, J. Nam, "A generic internet of things (IoT) platform supporting plug-and-play device management based on the semantic web," Journal of Ambient Intelligence and Humanized Computing, Sep. 2019.

[169] W. Yang, W. Liu, X. Wei, Z. Guo, K. Yang, H. Huang, L. Qi, "EdgeKeeper: A trusted edge computing framework for ubiquitous power internet of things," Front. Inf. Technol. Electron. Eng., vol. 22, no. 3, pp. 374–399, Jan. 2021.