information

MDPI

Article

An Application of SEMAR IoT Application Server Platform to Drone-Based Wall Inspection System Using AI Model

Yohanes Yohanie Fridelin Panduman [1], Radhiatul Husna [1], Noprianto [1], Nobuo Funabiki [1,*], Shunya Sakamaki [1], Sritrusta Sukaridhoto [2], Yan Watequlis Syaifudin [3] and Alfiandi Aulia Rahmadani [4]

[1] Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, Okayama 700-8530, Japan; p8f01q6f@s.okayama-u.ac.jp (Y.Y.F.P.); pwmn7i7q@s.okayama-u.ac.jp (R.H.); noprianto@s.okayama-u.ac.jp (N.); pdjm225d@s.okayama-u.ac.jp (S.S.)
[2] Department of Informatics and Computer, Politeknik Elektronika Negeri Surabaya, Surabaya 60111, Indonesia; dhoto@pens.ac.id
[3] Department of Information Technology, State Polytechnic of Malang, Malang 65141, Indonesia; qulis@polinema.ac.id
[4] Department of Electrical Engineering, State Polytechnic of Malang, Malang 65141, Indonesia; 1941170055@student.polinema.ac.id
* Correspondence: funabiki@okayama-u.ac.jp

Abstract: Recently, *artificial intelligence (AI)* has been adopted in a number of *Internet of Things (IoT)* application systems to enhance intelligence. We have developed a ready-made server with rich built-in functions to collect, process, display, analyze, and store data from various IoT devices, the *SEMAR (Smart Environmental Monitoring and Analytics in Real-Time)* IoT application server platform, in which various AI techniques have been implemented to enhance its capabilities. In this paper, we present an application of *SEMAR* to a drone-based wall inspection system using an object detection AI model called *You Only Look Once (YOLO)*. This system aims to detect *wall cracks* at high places using images taken via a camera on a flying drone. An edge computing device is installed to control the drone, sending the taken images through the *Kafka* system, storing them with the drone flight data, and sending the data to *SEMAR*. The images are analyzed via *YOLO* through *SEMAR*. For evaluations, we implemented the system using *Ryze Tello* for the drone and *Raspberry Pi* for the edge, and we evaluated the detection accuracy. The preliminary experiment results confirmed the effectiveness of the proposal.

Keywords: Internet of Things; AI; SEMAR; crack detection; drone; Kafka

Academic Editors: Eftim Zdravevski, Petre Lameski and Ivan Miguel Pires

Received: 27 November 2024
Revised: 11 January 2025
Accepted: 20 January 2025
Published: 24 January 2025

Citation: Panduman, Y.Y.F.; Husna, R.; Noprianto; Funabiki, N.; Sakamaki, S.; Sukaridhoto, S.; Syaifudin, Y.W.; Rahmadani, A.A. An Application of SEMAR IoT Application Server Platform to Drone-Based Wall Inspection System Using AI Model. *Information* **2025**, *16*, 91. https://doi.org/10.3390/info16020091

1. Introduction

Nowadays, the *Internet of Things (IoT)* has gained significant interest from both industrial and academic communities. With emerging device and networking technologies, IoT enables seamless interactions between the physical world and the digital world through the Internet [1]. It handles a variety of sensor devices and network connections across multiple domains [2]. As infrastructures for IoT application systems become ubiquitous, their success depends on abilities to collect, manage, and analyze data. Since an IoT application system needs to process a large volume of data in real time, efficient data handling is critical in designing and implementing it.

To enhance the capability of an IoT application system, *artificial intelligence (AI)* has been integrated [3,4]. AI has significantly gained popularity with its capabilities of processing a large volume of data with high accuracy. AI allows a system for learning data that can be operated autonomously and make intelligent decisions like human behaviors [5,6].

AI makes advanced sensor data analysis feasible by identifying data patterns, extracting valuable information, and making rapid decisions based on it [7]. These advantages will significantly enhance the potential of AI integrations across various IoT applications.

One potential application of this AI integration in an IoT application system is a building *wall inspection* system using drones. Due to their versatility, drones can expand the coverage area for wall monitoring. With the AI integration, it is expected that drones can autonomously fly and detect structural defects such as cracks in buildings.

Previously, we developed an IoT application server platform called *Smart Environmental Monitoring and Analytics in Real-Time (SEMAR)*. *SEMAR* is equipped with rich built-in functions for collecting, processing, displaying, analyzing, and storing data from various IoT devices [8]. Some functions are used for data analytics, aggregations, communications, and synchronizations in *Big Data* environments.

Moreover, *SEMAR* allows the addition of new *plug-in* functions through the *Representational State Transfer Application Programming Interface (REST API)* to access sensor data. In addition, AI models have been reviewed so that they can be implemented in *SEMAR* to enhance its capabilities [9]. Our review results show that AI capabilities can be implemented in an IoT platform through services such as *AI Model Management*, *Real-Time AI*, and *Batch AI* services.

Although *SEMAR* provides a robust platform for a lot of IoT application systems, the current implementation lacks functional supports for handling real-time processing with large amounts of data. This capability is essential for IoT applications such as drone-based surveillance systems.

In this paper, we present an application of *SEMAR* to a drone-based wall inspection system using an object detection AI model called *You Only Look Once (YOLO)* [10,11]. For this application, we trained the *YOLO* model and uploaded it to the *SEMAR* server through the function for *AI Model Management*. In our preliminary implementation, this system was designed to find wall cracks at high places using images taken via a camera on a flying drone. An edge computing device was installed to control the drone, transmit the images using the *Kafka* communication protocol from the drone to the edge, save them with the drone flight data in the edge, and also send the data to the *SEMAR* server. On the server, the *Real-Time AI* function continuously analyzes the stored images from the edge to detect cracks through an offline *YOLO* model. It stores the results in the database and can visualize them through user interfaces.

For evaluations of the proposal, we implemented the prototype system using *Ryze Tello* for the drone and *Raspberry Pi* for the edge, and we conducted experiments by running it to detect cracks at the #2 Engineering Building at Okayama University, Japan. Additionally, we compared performances between two data communication protocols, *Kafka* and *RabbitMQ*, in terms of efficiency and resource usage. Our experimental results confirmed the effectiveness of the implemented prototype system.

The rest of this paper is organized as follows: Section 2 presents related works. Section 3 briefly reviews our previous works on *SEMAR*. Section 4 presents the implementation of the AI functions in *SEMAR*. Section 5 presents the application of a drone-based wall crack detection system to *SEMAR*. Section 6 evaluates the application. Finally, Section 7 concludes this paper with a discussion of future works.

2. Related Works

In this section, we briefly present an overview of related works in the literature.

In [12], Munawar et al. reviewed methodologies for crack detection using image processing and machine learning approaches. Their findings indicate that the performance of image processing techniques is influenced by factors such as image resolution, illumina-

tion, and noise levels. Meanwhile, machine learning approaches have shown promising results, depending on the size of the dataset and the availability of computational resources. Furthermore, the authors highlighted that a key challenge in crack detection research lies in the development of systems capable of real-time detection.

In [13], Ali et al. highlighted the potential of deep learning methods for crack detection applications. They identified several key parameters to consider, including databases, processing hardware and software, network architecture, and balanced image datasets. The implementation of deep learning methods presents significant challenges due to the need for large amounts of memory and efficient processing devices. To address these challenges, it is necessary to optimize computational resources and algorithms to enable practical applications. For this purpose, our approach performs the detection process in the cloud server environment.

In [14], Su et al. evaluated the performance of the *YOLO* algorithm for vehicle-based crack detection in civil infrastructures. The research employed a vehicle-mounted camera and edge computing devices to predict cracks in real time. The system aims to improve road safety by reducing the risk of accidents caused by undetected cracks and improving overall driving conditions. The authors highlighted the feasibility of the *YOLO* model in providing accurate and rapid crack detection.

In [15], Yu et al. proposed the use of *YOLOv4* object detection models for the detection of bridge cracks using drones. The model was tailored to detect cracks in concrete structures, and it addressed the challenge of unbalanced positive and negative samples. To ensure suitability for real-time applications, the authors implemented several optimizations to improve both detection speed and performance.

In [16], Jung et al. presented the implementation of the *YOLO* model for object detection in drone images. The model was trained on a diverse dataset of drone images captured under different environmental conditions. This ensures robustness in various scenarios. However, the proposed system was not evaluated in a real-world environment. As a result, important aspects, such as data communication performance and the efficiency of the object detection process in practical applications, have not been evaluated.

In [17], Zhang introduced object detection algorithms for drone-captured images using the *YOLO* model. The authors improved the model by modifying its network structure. This improvement successfully addressed the challenges of detecting drone-captured images, including small targets, different scales, and complex backgrounds.

In [18], Kucukayan et al. proposed an indoor human detection system by integrating the *YOLO* algorithm with drone technology. Developed for Industry 4.0 applications, the system allows drones to monitor specific rooms or areas and detect small objects during surveillance. The approach relies on advanced image processing and AI algorithms to efficiently extract useful information from images. The study evaluated single-stage detectors such as the *YOLO* series. The results proved that the UAV-based object detection system applying *YOLOv7* performs well for real-time applications on edge devices. It demonstrated that the model can effectively detect small objects, even at a resolution of 416 × 416, making it suitable for limited-resource environments.

In [19], Patel et al. analyzed the performance of data communication services for video streaming applications in cyber–physical systems. These services were designed to transfer video frames from the source to a cloud server and then to the end user. The experimental results showed that both *RabbitMQ* and *Kafka* communication protocols are suitable for this use case. *Kafka* was able to send video frames as JSON objects to multiple consumers while maintaining the order of messages. On the other hand, *RabbitMQ* ensured message delivery but did not preserve the order, resulting in lower fault tolerance compared to *Kafka*.

To address this limitation, additional methods are required to effectively manage the order of messages.

In [20], Liao et al. proposed a drone-based marine trash detection system that used the *YOLO* model for efficient trash detection. The system worked by collecting aerial images of coastal areas. The drone was equipped with an embedded system to perform trash detection using the *YOLO* model. The detection results were transmitted to a server via a Kafka communication service and stored in a database. Although this approach shares similarities with our proposed system, our system provides a more versatile and dynamic solution by allowing users to flexibly manage the AI model, supporting not only *YOLO* but also various other object detection techniques based on their specific requirements.

In [21], Karpiuk et al. implemented *Kafka* communication to support near-real-time image analysis. *Kafka* was utilized to facilitate the data flow across a cluster of nodes. This approach reduced the risk of data loss and ensured fault tolerance, even under heavy workloads. The authors highlighted key parameters for achieving optimal performance in real-world scenarios, such as image resolution, image complexity, computational resources, hardware constraints, and network latency.

3. Review of *SEMAR* IoT Application Server Platform

In this section, we review the *SEMAR* IoT application server platform.

3.1. System Overview

We have developed an IoT application server platform for integrating independent IoT application systems called *SEMAR* [8]. Figure 1 shows the system overview of *SEMAR*. It offers rich *built-in* functions for data analysis, processing, communications, synchronization, and visualization help deployments of IoT application systems. The functions in *SEMAR* are grouped into *data input*, *data processing*, and *data output* components, which are managed via the *management* system.

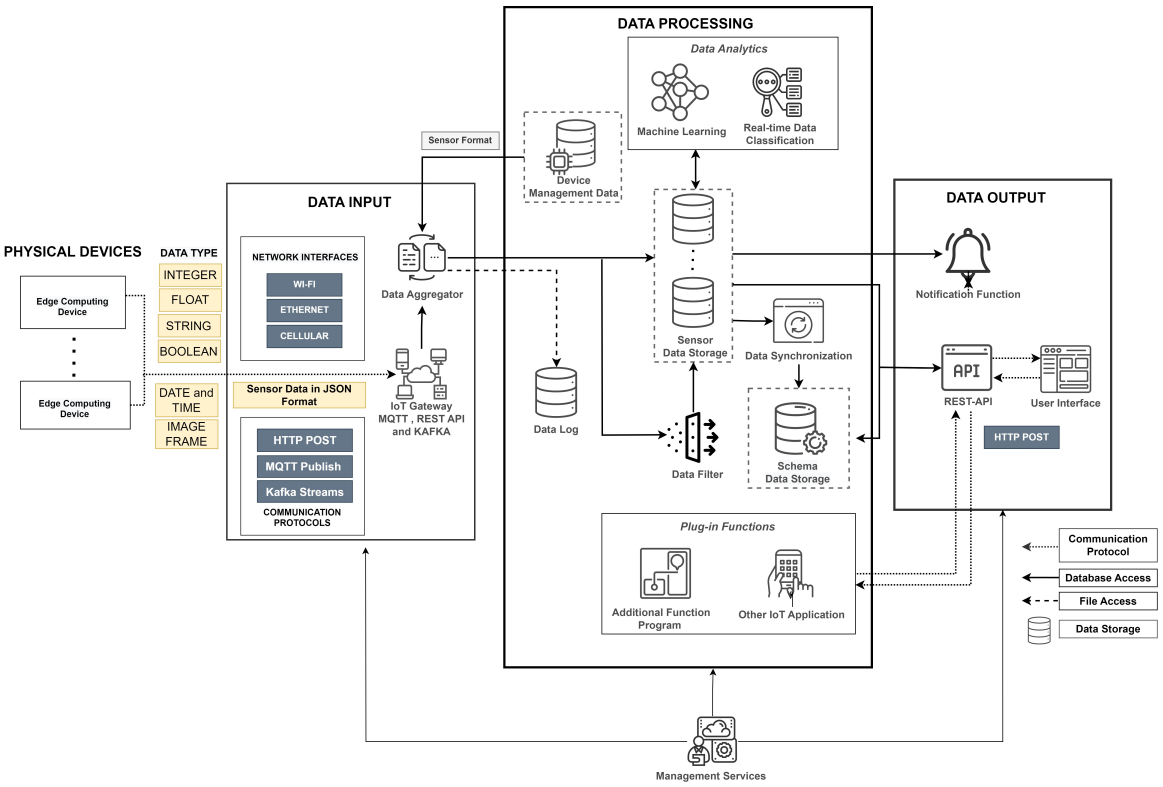

Figure 1. Design Overview of *SEMAR* IoT application server platform.

3.2. Data Communications

Through *MQTT* and *HTTP-POST* communication protocols, the *data input* group accepts the receipt of data from IoT devices in the *JavaScript Object Notation (JSON)* format. When *SEMAR* receives data, the *data aggregator* function converts them into a consumable format, extracts the necessary information, and stores the result in the *MongoDB* database [22]. The *data processing* group provides the *data filter* and *data synchronization* functions. The *data filter* function reduces noise and inaccuracy in data using digital filtering techniques. The *data synchronization* function integrates the data from different resources into a single data record. The *data output* group prepares the web-based *user interface* to visualize the data. These functions can be used without modifying their source codes. Additionally, the *REST API* service is provided to facilitate data sharing and integration with *plug-in* functions or other systems, using *HTTP POST* communications in the JSON format.

3.3. Edge Device Framework

In [23], we introduced the *edge device framework* to enhance device utilizations by allowing the remote configuration of the edge through *SEMAR* under the device management function. It allows the creation, updating, and deletion of the edge configuration file by accessing the functions through the *user interface*.

4. Implementation of AI Functions in *SEMAR*

In this section, we present an implementation of AI functions in *SEMAR*.

4.1. Implementation Overview

Figure 2 illustrates the system overview of implementations of AI functions in *SEMAR*. The system consists of *AI Model Management*, *Real-Time AI*, and *Batch AI* functions. The system allows users to generate AI models outside the environment of the *SEMAR* platform using collected sensor data as datasets. The *AI Model Management* function manages the AI models generated by users for deployments in the system. The *Real-Time AI* function provides a data processing capability using AI in real-time scenarios. The *Batch AI* function allows data processing using existing data in the database through AI models. In addition, by integrating the *Edge Device Framework* [23], AI models can also be implemented on edge computing devices.

4.2. AI Model Management

The implementation of an AI model requires a structured approach. It includes defining goals, collecting datasets, building models, and deploying them. *SEMAR* should be designed to facilitate the integration of AI models into applications. Then, we implement the *AI Model Management* function to manage, add, remove, and deploy AI models through the interface.

This function allows the generation of AI models outside of *SEMAR*. First, a user accesses to sensor data stored in *SEMAR* and downloads it for the data training process of the AI model. Then, the generated AI model is uploaded to *SEMAR* through the interface. Here, it is necessary to specify the model information, such as the name, version, inputs, outputs, and type of AI model. Once the AI model is registered, the function stores it in storage.

4.3. Real-Time and Batch AI Processing

Two services are considered to perform data processing using an AI model, namely the *Real-Time AI* service and the *Batch AI* service.

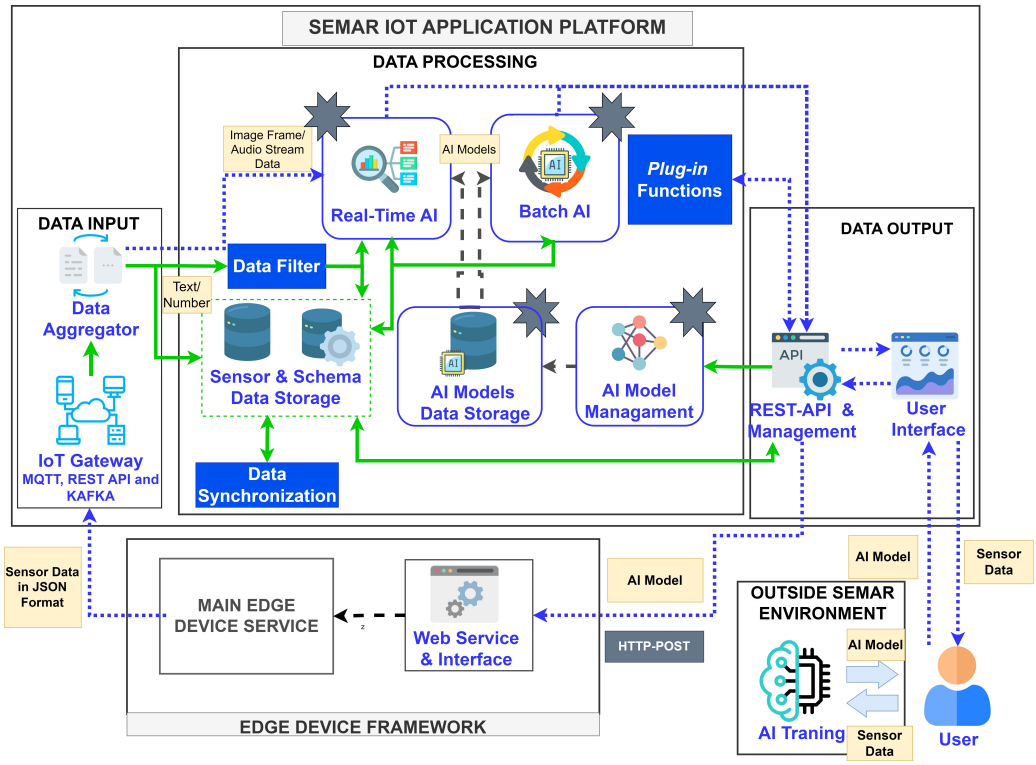

Figure 2. Design overview of AI techniques in *SEMAR* IoT application server platform.

4.3.1. Real-Time AI Processing

The *Real-Time AI* service is designed for IoT applications that require immediate AI processing. It connects the *IoT cloud gateway* and the *data aggregator* to enable efficient data stream processing. When the *IoT cloud gateway* receives data, the *data aggregator* verifies the data format in standards such as *JPEG*, *WAV*, text, or numeric. It then sends the data to the *Real-Time AI* service through the *Kafka* communication protocol for image frames or audio data. Finally, the processed results are stored in the database.

4.3.2. Batch AI Processing

The *Batch AI* service is designed to process AI tasks on existing data stored in the system. This service can handle large datasets that cannot be processed in real-time scenarios. Unlike *Real-Time AI* services that process data automatically, for this service, it is necessary to manually select specific data that will be processed through the user interface and select the appropriate AI model to perform processing. The platform systematically applies the AI model to the selected data, produces the results, and saves them in the database. By incorporating the *Real-Time AI* and *Batch AI* services, *SEMAR* provides flexible solutions for both immediate processing and post-processing tasks.

4.4. Implementation in Edge Device

In [23], we proposed an extension of *SEMAR* to the *edge device framework*. This framework allows the remote configuration of the edge device from the server. The framework offers capabilities for connecting to multiple IoT sensors with standardized formats for data processing and providing various output mechanisms for utilizing the collected data.

In this study, we enhance the data processing component by integrating lightweight AI models on edge devices. Figure 2 shows the process of obtaining the AI model from the *SEMAR* server through *HTTP-POST* communication. Then, the *main service* of the framework reads the AI model and initiates the collection of sensor data. After the sensor data are received, the framework will process the AI model.

5. Application for Drone-Based Wall Inspection

In this section, we present the application of *SEMAR* with the AI implementation to a drone-based wall inspection system.

5.1. System Overview

Figure 3 illustrates the system overview of the application for the drone-based wall inspection. The IoT gateway using the *Kafka* communication protocol, the *Real-Time AI* function for wall crack detections, data storage, and user interfaces are implemented in *SEMAR*.

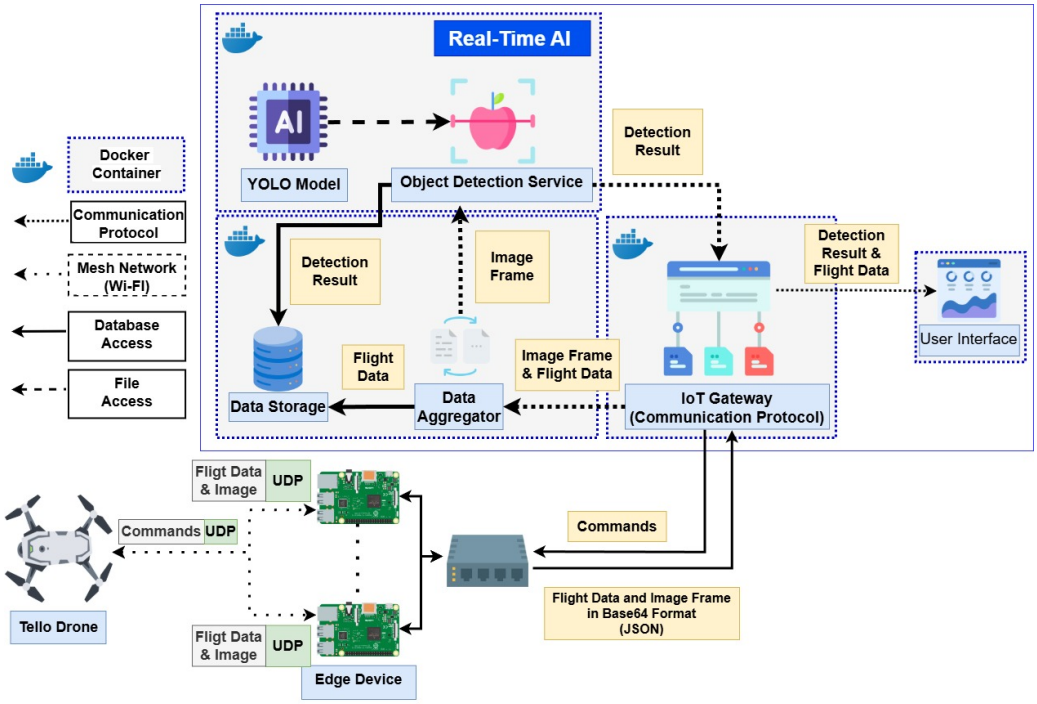

Figure 3. System overview of drone-based wall inspection system.

We built the system based on a microservices architecture [24] to simplify the processes involved in software development and operation. This approach divides the system into independent small services. *Docker* [25] was used to package each service and its dependencies into *Docker containers*. It can prevent compatibility issues between different environments, as *Docker containers* contain all the necessary dependencies to run the service. It also allows us to scale the system more easily, as each service can be deployed and managed independently.

5.2. Drones and Edge Devices

Flying drones collect the image data around the target building. Edge devices are placed around the building to control the drones and receive data from them. A mesh network is implemented for connecting the drones flying around the field. A drone transmits the flight data and images to an edge device through UDP communication over a Wi-Fi connection. Once an edge device receives data, it resizes the image frames into 612×460 pixels and sends them to the *SEMAR* server over ethernet connections.

5.3. Communication Protocol

Previously, the *IoT gateway* only supported *MQTT* and HTTP-POST communication protocols for receiving sensor data. However, these protocols are limited in the amount of data that they can transfer. To handle large data, we adopt the *Kafka* [26] and *RabbitMQ* [27]

protocols in the *IoT Gateway*. *Kafka* is the communication protocol designed for image streaming applications [28]. Like *MQTT*, each message in *Kafka* is identified by a topic. A *Kafka* broker receives messages from producers and distributes them to consumers by the topic. For this purpose, the *Confluent Kafka* is utilized in this research.

One key strength of *Kafka* is its scalability. It is able to scale by adding more brokers to the cluster, allowing it to handle large amounts of data from multiple producers and consumers. In addition, it provides disk-based retention to ensure that messages are reliably stored for a set period of time. Thus, it prevents data loss due to system problems. With these features, *Kafka* is well suited to managing large data streams and supporting near-real-time processing, which is essential for applications such as image analysis and streaming.

RabbitMQ is an open-source message broker that acts as middleware for various applications. This service is built in the *Erlang* programming language, and it implements the *Advanced Message Queuing Protocol (AMQP)* for reliable message delivery and asynchronous communication. A *RabbitMQ* broker operates with two primary components, including *Exchanges* and *Queues*. *Exchanges* are responsible for determining how messages are routed and distributed, and they receive messages from producers and forward them to one or more queues according to routing rules. *Queues* act as buffers to temporarily store messages until consumers are ready to receive them.

Each message in *RabbitMQ* is identified by a routing key, which helps the exchange decide how to route the message to the matching queue based on its type and routing rules. This allows for flexible message routing. In addition, *RabbitMQ* is designed with routing flexibility, clustering, and message tracking capabilities. This provides a reliable and scalable solution, particularly for distributed systems where different components need to communicate asynchronously.

The *data aggregator* for the *Kafka* and *RabbitMQ* communication protocols is also implemented. It receives the flight data and the captured images from an edge device. Then, it transmits them into the *Real-Time AI* function and data storage. It visualizes the results of the *Real-Time AI* AI function in the user interface.

5.4. Real-Time AI Functions for Wall Crack Detection

According to [9], the *YOLO* algorithm has gained popularity for its efficiency in real-time or near-real-time scenarios. Therefore, we leveraged the *YOLO* algorithm for object detection in the *Real-Time AI* function. For this purpose, the *YOLO* model was trained using the open-source crack datasets [29]. Then, it was uploaded to the *SEMAR* server, and it was deployed in the *Real-Time AI* function to perform crack detection.

The purpose of this IoT application system is to continuously detect cracks within images captured via drones, which is achieved through two processes, as shown in Figure 4. The first process receives the image frames from the *data aggregator* and organizes them into a sequence queue. The second process performs the object detection on each image frame using the trained *YOLO* model. When a cracked object is detected in an image, the system stores the results in the data storage. Both processes are performed in parallel, ensuring that the functions continue to receive data without interruption. This approach aims to reduce processing time by enabling continuous data flow.

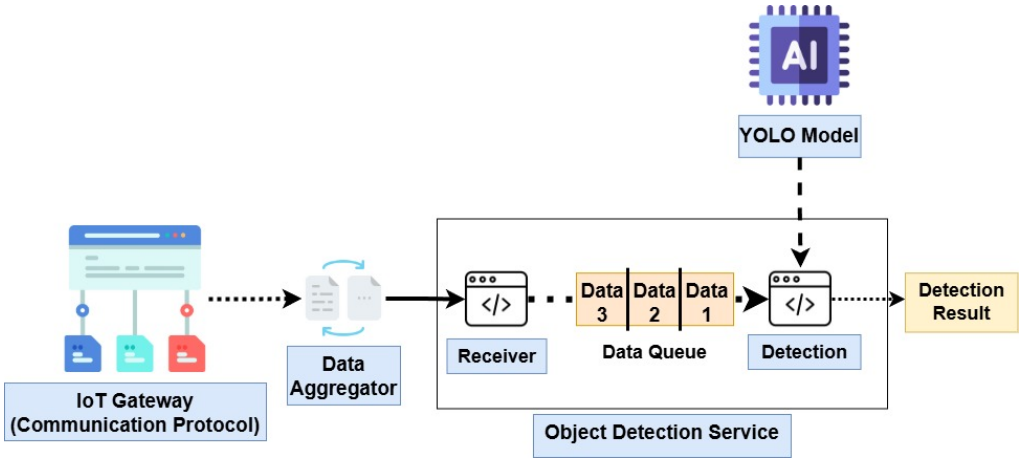

Figure 4. Flow of *Real-Time AI* function for detecting wall cracks.

5.5. Data Storage and User Interface

For the flight data, the *data aggregator* retrieves the information from the IoT gateway and stores it in a *MongoDB* database. The image frames are saved in the data storage. The *user interface* displays the crack detection results and the flight data.

6. Evaluation

In this section, we evaluate the application of the *SEMAR* IoT application platform to the drone-based wall inspection system.

6.1. Experimental Scenario

In this study, we evaluated the performance of communication protocols and the wall crack detection process. First, we compared the performance of *Kafka* and *RabbitMQ* in transmitting image frames in various conditions. The experiments focused on investigating the transmission time and system resource utilization, including CPU and memory (RAM) usage, while transmitting data from the drone at different frame rates. These parameters were selected to assess the efficiency and stability of communication protocol services in managing image data transmission under different scenarios, ranging from low to high frame rates.

For this purpose, the preliminary experiments were conducted in the #2 Engineering Building of Okayama University. We utilized *Ryze Tello* for the drone and *Raspberry Pi* for the edge device. The *Ryze Tello drone* flew inside this building to capture wall images and sent them to the edge device. The edge device converted the image into 613 × 460 pixels and sent it into the *SEMAR* server through communication protocols at 1, 2, 4, 10, and 12 *frames per second (FPS)*. Table 1 presents the server specification in the experiments.

Table 1. Device and software specifications for *SEMAR* server.

Items	Specifications
CPU	Intel(R) Xeon(R) Gold 5218
CPU Cores	8
Memory	24 GB
Operating System	Ubuntu 20.04.5
GPU	NVIDIA Quadro RTX 6000
Manufacturer	Hewlett Packard Enterprise, based in Spring, TX, USA

In order to retrieve information about server resources, *Prometheus* was used with *Node Exporter* as its key component [30,31]. *Node Exporter* collects data on various server

resources, including CPU usage, memory, and disk and network activity. Then, it transmits these data to *Prometheus* as an interface to visualize the data.

The second experiment aimed to evaluate the performance of the wall crack detection model. In this study, we employed the *YOLOv7* object detection algorithm to identify wall cracks in images captured via drones. The *YOLOv7* model consists of three main components, including the backbone network, the bottleneck layer, and the detection network [32]. The backbone network primarily performs feature extraction. It is composed of standard convolutional layers, max-pooling layers, and *Extended Efficient Layer Aggregation Network (ELAN)* modules to generate rich feature maps. The bottleneck layer refines these extracted features to improve the object detection performance. Finally, the detection network acts as a classifier and regressor, providing the final predictions including the bounding box coordinates, class probabilities, and confidence scores for the detected objects. To ensure consistency in feature extractions, each image in the dataset was resized to 416×416 pixels. This resizing not only standardizes the input dimensions for the model but also reduces the computational complexity. Thus, it allows for more efficient processing during both training and detection (recognition).

The standard evaluation of a *YOLOv7* model is measured by the values of *recall*, *precision*, *F1-score*, and *mean average precision (mAP)* [33]. In this study, we used these metrics to evaluate the performance of the *YOLOv7* model in detecting cracks in images. *Recall* represents the performance of the model to detect all the relevant objects in the validation image. *Precision* refers to the ability of the model to identify the proportion of detected objects that are correct over the ground truth. The *F1-score* evaluates the first harmonic mean between *recall* and *precision*. The *mAP* metric captures the accuracy and coverage of the object detection bounding box.

In the object detection context, the *intersection over union (IoU)* is used to validate the detected objects against the ground truth objects [34]. This method identifies a detected object as a correct one if the *IoU* between the detected object's bounding box and the ground truth bounding box meets a specified threshold ($IoU > threshold$). In this study, we considered a threshold of 50%.

To further evaluate the model's performance, we also validated it by applying the *Box Loss* validation method [35]. This method measures the difference in location and size between the detected bounding box and the ground truth bounding box. It offers an important indication of how well the model can accurately locate cracks in images captured via the drone. We used a dataset consisting of 5704 crack images for training and 1427 images for validation, and we trained the model for 500 epochs.

In addition, we also measured the time required for the model to detect cracks in drone-captured images. After training the model, we uploaded it to the *SEMAR* server and deployed it in the *Real-Time* function of the server for continuous crack detections. After experiments, we measured the average inference time by using the time between one instance of image reception from the communication service at the model and the detection result output.

6.2. Communication Protocol Results

Figure 5 shows the average time that the *Kafka* and *RabbitMQ* communication protocols needed to transmit the image frames from the edge device to the *SEMAR* server until the server received the final ones. It shows that *Kafka* and *RabbitMQ* achieve quite similar performances at any fps rate. In both protocols, the required processing and transmission time becomes longer as the fps rate increases. At $5fps$, the average transmission time increases significantly compared at $2fps$, at which *Kafka* required 38.27 s, and *RabbitMQ*

required slightly longer at 39.45 s. This finding demonstrates that *Kafka* performs slightly better at this medium workload.

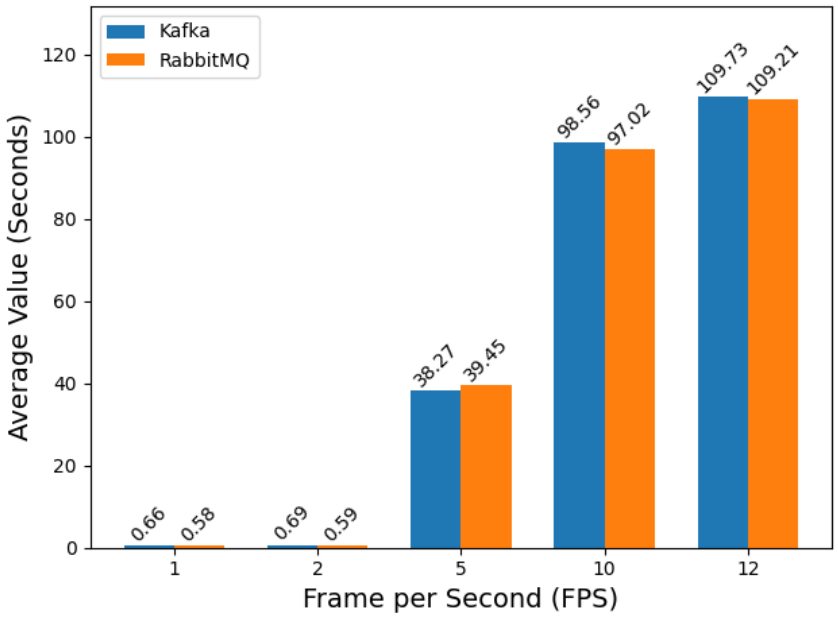

Figure 5. Comparison of average total transmission time at different fps.

Figure 6 illustrates the analysis of RAM usage at different fps levels for the *Kafka* and *RabbitMQ* communication protocols. The results show that *Kafka* consistently consumes significantly more RAM than *RabbitMQ*, with differences up to 1.6 GB. This is due to the architecture of *Kafka*, which is built on the *Java Virtual Machine (JVM)* and consequently requires more memory resources to operate.

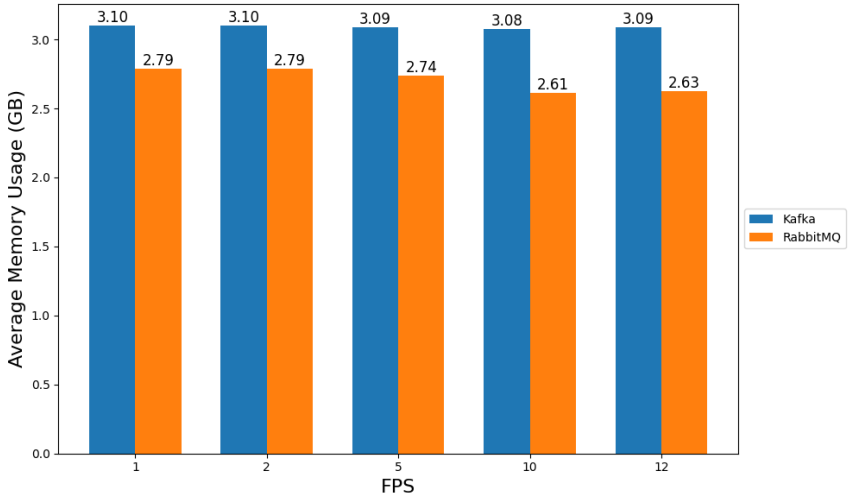

Figure 6. Average RAM use of *Kafka* and *RabbitMQ* communication protocols.

An analysis of the CPU usage was also conducted during the experiments. Figure 7 presents the usage rate of each CPU core at the fps rates of 1, 2, 5, 10, and 12. The results obtained at $1fps$, $2fps$, and $5fps$ indicate that the CPU usage rate for the *Kafka* and *RabbitMQ* communication protocols are similar. However, the results at $10fps$ and $12fps$ show that the CPU usage rate for *Kafka* is slightly lower compared to *RabbitMQ*. This indicates that *Kafka* demands lower processing power at higher frame rates and offers stronger features and durability than *RabbitMQ*. Its efficient performance at higher frame

rates demonstrates the better scalability and responsiveness of *Kafka*, making it more suited for applications requiring frequent data transmissions.

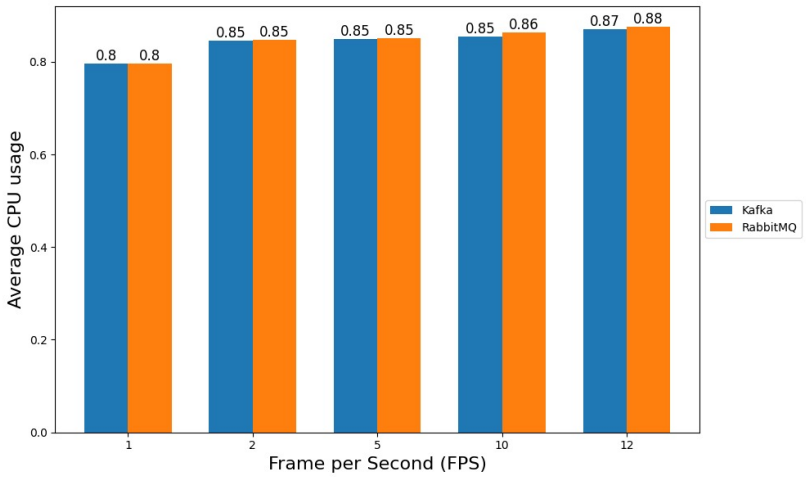

Figure 7. Comparison of average CPU usage rate for communication protocols.

6.3. Wall Crack Detection Results

Next, we assessed wall crack detection results. First, the *YOLO model* was evaluated by using standard four metrics for object detection evaluations. The metrics consist of *recall*, *precision*, *F1-score*, and *mAP*. To calculate these metrics, the results of *true positives*, *false negatives*, and *false positives* are necessary. *True positives* represent the correctly detected objects that passed the validation process based on the *IoU* threshold with the ground truth. *False negatives* represent the ground truth objects that were not detected. *False positives* refer to the objects that were incorrectly detected.

The calculation formulas for *recall* and *precision* are given as follows:

$$Recall = \frac{T_p}{T_p + F_p} \tag{1}$$

$$Precision = \frac{T_p}{T_p + F_n} \tag{2}$$

where T_p, F_p, and F_n represent the number of *true positive* cracks, the number of *false positive* cracks, and the number of *false negative* cracks, respectively.

Then, the *F1-score* is calculated as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

Our evaluation results show that the adopted YOLO model correctly identified 1683 *true positive* cracks, 181 *false negative* ones, and 125 *false positive* ones from the 1427 validation images. Therefore, the *precision* of the model is 0.93, the *recall* is 0.90, and the *F1-score* is 0.91. These results indicate that the adopted YOLO model is reliable for this crack detection application with high accuracy and completeness. Furthermore, the *mAP* of this generated model is 0.91, which demonstrates that the model can accurately and consistently detect crack objects in given images.

Figure 8 shows the results of the *Box Loss* validation method obtained during the training process. It achieved 0.012 for the final validation and 0.011 for the training, respectively. These results indicate that the model performs well in both the training and validation phases. The small difference between the training and validation box losses indicates that this model can perform well for new images.

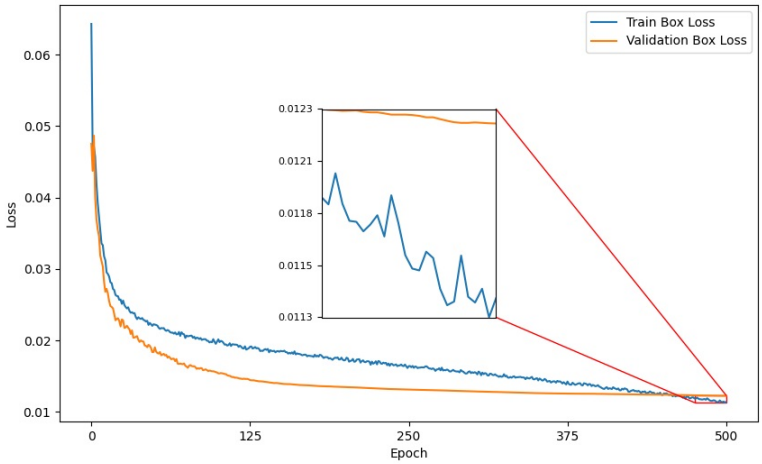

Figure 8. *Box Loss* validation results of generated model.

To evaluate the proposed system, one *Ryze Tello* drone was controlled to capture images of wall cracks. Figure 9 illustrates a position of the drone while capturing the image of wall cracks during the experiment. The drone continuously transmitted the images to the edge computing device using *Raspberry Pi*. This edge device sent the data to the *SEMAR* server through the *Kafka* communication protocol.

In the server, the *Real-Time AI* function analyzes the images transmitted from the edge device. When this function receives the images captured via the drone, it detects the cracks by applying the trained *YOLO* model to them. Finally, the data including the detecting results are stored in the data storage in the server. They can be visualized through a web browser for the user interface.

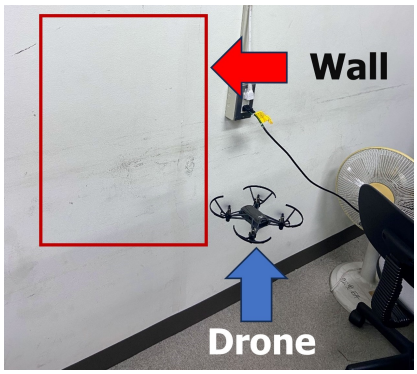

Figure 9. Photo of drone capturing image of wall cracks.

Figure 10 shows the *user interface* in a browser to show the captured image and the crack detection result. In the detection result image, the bounding box indicates the location of the detected crack, the crack is highlighted, and the class label ("crack") and the confidence score (0.81) are denoted.

Figure 11 illustrates the computation time of the *Real-Time* AI function in detecting crack images, for which the average time was 857 ms. Therefore, the use of the *YOLOv7* algorithm with the drone-captured images can be promising for wall drone-based inspection systems.

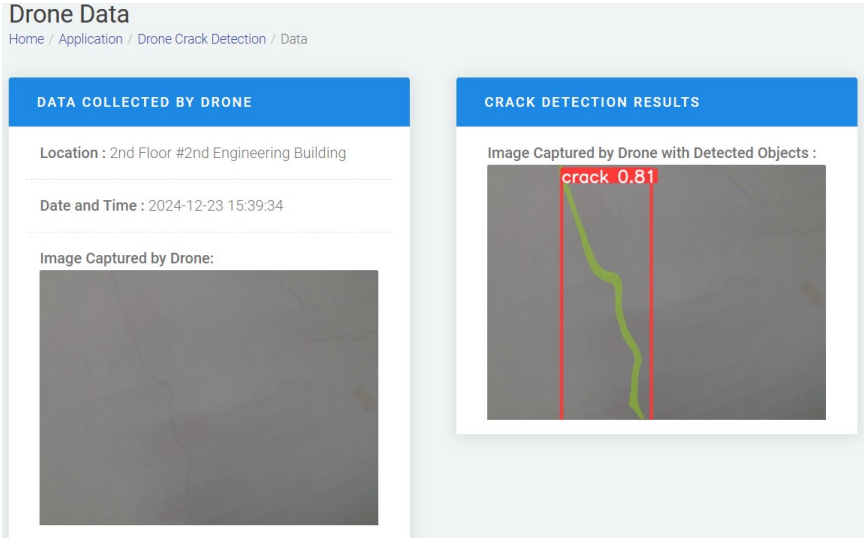

Figure 10. User interface for detection of *SEMAR* server.

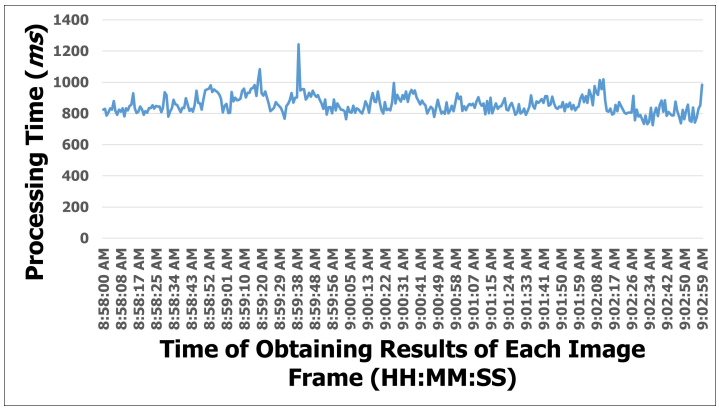

Figure 11. Computation time of crack detection.

6.4. Comparison with Traditional Inspection Methods

To further support our evaluations, we compared our proposed system with traditional inspection methods, including manual visual inspections and conventional image processing techniques. The manual visual inspection method typically requires auxiliary tools, such as manned lifts and aerial platforms [36]. This approach is associated with a high cost, low efficiency, and significant risks, which can limit its dependability for building inspections [37]. In addition, the accuracy of this method can be compromised due to its reliance on the individual's ability to detect wall cracks.

In contrast, conventional imaging techniques will offer higher efficiency and lower risks compared to a manual visual inspection method. They can include *infrared (IR)* image-based processing techniques. The *IR* image-based processing technique utilizes temperature differences to identify structural defects in various materials [38]. In [39], Rodríguez-Martín et al. utilized an IR camera to detect cracks with this geometric characterization and orientation. This camera can detect temperature differences on the surfaces of exterior walls, which can indicate the presence of cracks.

Furthermore, In [40], Ivan et al. presented an implementation of an IR camera for infrastructure inspections. The proposed system worked by capturing thermal images of a structure using an infrared camera, which can reveal temperature differences that may not be visible to the naked eye. Despite this advantage, the adopted technique is highly dependent on various external environmental factors, such as temperature and humidity.

As a result, it often requires manual thresholding and segmentations to remove noise that may be present in thermal data.

Compared to these traditional methods, the drone-based wall inspection system will offer a more cost-effective and efficient approach. By using drones, it can eliminate the need for costly and hazardous equipment such as manned lifts. In addition, the integration of AI-based analysis enables automated and accurate crack detection, reducing the reliance on human judgments and minimizing errors.

7. Conclusions

This paper has presented an application of the *SEMAR* IoT application server platform to a drone-based wall inspection system using the *YOLO* model. It utilizes a flying drone to take images of wall cracks at high places in a building. An edge computing device is installed to control the drone, receive image frames from it, and send them along with the drone flight data to the *SEMAR* server through the communication protocol. The *Real-Time* AI function using the *YOLO model* detects cracks from the image frames continuously, for which the results are stored in the *Mongo* database and visualized through the user interface. A prototype system was implemented using *Ryze Tello* for the drone and *Raspberry Pi* for the edge computing device. The application and evaluation results validated the feasibility of the proposal.

In future works, we will continue to develop new functions and services in *SEMAR*, including other AI models, and apply it to various IoT application systems. Furthermore, we will implement multi-cluster and multi-node configurations to improve the efficiency of distributing image frames on the *SEMAR* server. This approach aims to optimize the performance of the messaging protocols.

Author Contributions: Conceptualization, Y.Y.F.P., R.H., N.F. and S.S. (Sritrusta Sukaridhoto); methodology, Y.Y.F.P., N. and R.H.; software, Y.Y.F.P., S.S. (Shunya Sakamaki) and N.; writing—original draft preparation, Y.Y.F.P.; writing—review and editing, N.F.; validation, Y.W.S. and A.A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors thank the reviewers for their thorough reading and helpful comments.

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

References

1. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
2. Stankovic, J.A. Research Directions for the Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 3–9. [CrossRef]
3. Alahi, M.E.; Sukkuea, A.; Tina, F.W.; Nag, A.; Kurdthongmee, W.; Suwannarat, K.; Mukhopadhyay, S.C. Integration of IoT-enabled technologies and Artificial Intelligence (AI) for Smart City Scenario: Recent advancements and future trends. *Sensors* **2023**, *23*, 5206. [CrossRef]
4. Sharma, K.; Shivandu, S.K. Integrating Artificial Intelligence and internet of things (IoT) for enhanced crop monitoring and management in Precision Agriculture. *Sens. Int.* **2024**, *5*, 100292. [CrossRef]
5. Duan, Y.; Edwards, J.S.; Dwivedi, Y.K. Artificial Intelligence for Decision Making in the Era of Big Data – Evolution, Challenges and Research Agenda. *Int. J. Inf. Manag.* **2019**, *48*, 63–71. [CrossRef]
6. Belgaum, M.R.; Alansari, Z.; Musa, S.; Mansoor Alam, M.; Mazliham, M.S. Role of Artificial Intelligence in Cloud Computing, IoT and SDN: Reliability and Scalability Issues. *Int. J. Electr. Comput. Eng. (IJECE)* **2021**, *11*, 4458. [CrossRef]

7. Saleem, T.J.; Chishti, M.A. Deep Learning for the Internet of Things: Potential Benefits and Use-cases. *Digit. Commun. Netw.* **2021**, *7*, 526–542. [CrossRef]
8. Panduman, Y.Y.F.; Funabiki, N.; Puspitaningayu, P.; Kuribayashi, M.; Sukaridhoto, S.; Kao, W.-C. Design and Implementation of SEMAR IoT Server Platform with Applications. *Sensors* **2022**, *22*, 6436. [CrossRef] [PubMed]
9. Panduman, Y.Y.; Funabiki, N.; Fajrianti, E.D.; Fang, S.; Sukaridhoto, S. 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* **2024**, *15*, 153. [CrossRef]
10. Zheng, G. YOLOX: Exceeding YOLO Series in 2021. *arXiv* **2021**, arXiv:2107.08430v2. [CrossRef]
11. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv* **2022**, arXiv:2207.02696
12. Munawar, H.S.; Hammad, A.W.; Haddad, A.; Soares, C.A.; Waller, S.T. Image-based crack detection methods: A review. *Infrastructures* **2021**, *6*, 115. [CrossRef]
13. Ali, R.; Chuah, J.H.; Talip, M.S.; Mokhtar, N.; Shoaib, M.A. Structural crack detection using deep convolutional neural networks. *Autom. Constr.* **2022**, *133*, 103989. [CrossRef]
14. Su, P.; Han, H.; Liu, M.; Yang, T.; Liu, S. Mod-Yolo: Rethinking the Yolo Architecture at the level of feature information and applying it to crack detection. *Expert Syst. Appl.* **2024**, *237*, 121346. [CrossRef]
15. Yu, Z.; Shen, Y.; Shen, C. A real-time detection approach for bridge cracks based on YOLOv4-FPM. *Autom. Constr.* **2021**, *122*, 103514. [CrossRef]
16. Jung, H.-K.; Choi, G.-S. Improved Yolov5: Efficient object detection using drone images under various conditions. *Appl. Sci.* **2022**, *12*, 7255. [CrossRef]
17. Zhang, Z. Drone-yolo: An efficient neural network method for target detection in drone images. *Drones* **2023**, *7*, 526. [CrossRef]
18. Kucukayan, G.; Karacan, H. Yolo-IHD: Improved real-time human detection system for indoor drones. *Sensors* **2024**, *24*, 922. [CrossRef]
19. Patel, U.; Tanwar, S.; Nair, A. Performance analysis of video on-demand and live video streaming using cloud based services. *Scalable Comput. Pract. Exp.* **2020**, *21*, 479–496. [CrossRef]
20. Liao, Y.-H.; Juang, J.-G. Real-time UAV trash monitoring system. *Appl. Sci.* **2022**, *12*, 1838. [CrossRef]
21. Karpiuk, N.; Klym, H.; Tkachuk, T. Usage of apache kafka for low-latency image processing. *Electron. Inf. Technol.* **2024**, *26*, 46–58.
22. MongoDB. Mongodb: The Application Data Platform. Available online: https://www.mongodb.com/ (accessed on 21 November 2024).
23. Panduman, Y.Y.F.; Funabiki, N.; Ito, S.; Husna, R.; Kuribayashi, M.; Okayasu, M.; Shimazu, J.; Sukaridhoto, S. An Edge Device Framework in SEMAR IoT Application Server Platform. *Information* **2023**, *14*, 312. [CrossRef]
24. Bixio, L.; Delzanno, G.; Rebora, S.; Rulli, M. A flexible IoT stream processing architecture based on microservices. *Information* **2020**, *11*, 565. [CrossRef]
25. Docker. Available online: https://docs.docker.com/get-started/get-docker/ (accessed on 21 November 2024).
26. Kreps, J.; Narkhede, N.; Rao, J. Kafka: A distributed messaging system for log processing. In Proceedings of the 6th International Workshop on Networking Meets Databases, Athens, Greece, 12–16 June 2011; Volume 11, pp. 1–7.
27. Dixit, S.; Madhu, M. Distributing messages using rabbitmq with advanced message exchanges. *Int. J. Res. Stud. Comput. Sci. Eng.* **2019**, *6*, 24–28.
28. Htut, A.M.; Aswakul, C. Development of near real-time wireless image sequence streaming cloud using Apache Kafka for Road Traffic Monitoring Application. *PLoS ONE* **2022**, *17*, e0264923. [CrossRef]
29. University, "Crack Instance Segmentation Dataset (V2) by University," Roboflow. Available online: https://universe.roboflow.com/university-bswxt/crack-bphdr/dataset/2 (accessed on 22 February 2024).
30. Yudha Erian Saputra, M.; Noprianto; Noor Arief, S.; Nur Wijayaningrum, V.; Syaifudin, Y.W. Real-time server monitoring and notification system with prometheus, Grafana, and telegram integration. In Proceedings of the 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS), Manama, Bahrain, 28–29 January 2024; pp. 1808–1813. [CrossRef]
31. Malhotra, R.; Bansal, A.; Kessentini, M. Deployment and performance monitoring of Docker based Federated Learning Framework for software defect prediction. *Clust. Comput.* **2024**, *27*, 6039–6057. [CrossRef]
32. Zeng, Y.; Zhang, T.; He, W.; Zhang, Z. YOLOv7-UAV: An Unmanned Aerial Vehicle Image Object Detection Algorithm Based on Improved YOLOv7. *Electronics* **2023**, *12*, 3141. [CrossRef]
33. Padilla, R.; Netto, S.L.; da Silva, E.A.B. A Survey on Performance Metrics for Object-Detection Algorithms. In Proceedings of the 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), Niterói, Brazil, 1–3 July 2020; pp. 237–242.
34. Borui, J.; Ruixuan, L.; Jiayuan, M.; Tete, X.; Yuning, J. Acquisition of Localization Confidence for Accurate Object Detection. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 784–799. [CrossRef]

35. Kasper-Eulaers, M.; Hahn, N.; Berger, S.; Sebulonsen, T.; Myrland; Kummervold, P.E. Short Communication: Detecting Heavy Goods Vehicles in Rest Areas in Winter Conditions Using YOLOv5. *Algorithms* **2021**, *14*, 114. [CrossRef]
36. Vossoughi, H.; Siddiqui, R.I. Industrial rope access—An alternate means for inspection, maintenance, and repair of building facades and structures. In *STP1444-EB Building Facade Maintenance, Repair, and Inspection*; ASTM International: West Conshohocken, PA, USA, 2004.
37. Jung, S.; Song, S.; Youn, P.; Myung, H. Multi-Layer Coverage Path Planner for Autonomous Structural Inspection of High-Rise Structures. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 1–9.
38. Karpf, A.; Selig, M.; Alchaar, A.; Iskander, M. Detection of cracks in concrete using near-IR fluorescence imaging. *Sci. Rep.* **2023**, *13*, 18880. [CrossRef] [PubMed]
39. Rodríguez-Martín, M.; Lagüela, S.; González-Aguilera, D.; Martínez, J. Thermographic test for the geometric characterization of cracks in welding using IR image rectification. *Autom. Constr.* **2016**, *61*, 58–65. [CrossRef]
40. Ivan, G.; Susana, L.; Pedro, A. Infrared Thermography's Application to Infrastructure Inspections. *Infrastructures* **2018**, *3*, 35. [CrossRef]