National College of Ireland

Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems

MSc Research Project
Cloud Computing

Olajide Olawuyi
Student ID: x23214619

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Olajide Olawuyi
Student ID:	x23214619
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	13/12/2024
Project Title:	Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems
Word Count:	XXX
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems

Olajide Olawuyi
x23214619

github link: https://github.com/Olajide1/msc-project.git

Abstract

Historically, communities have faced devastating fire hazards, underscoring the need for early detection and alert systems to ensure effective evacuation planning. This study integrates wireless sensor networks (WSNs), cloud computing, and machine learning to develop a robust fire emergency management system. The system employs DHT11 and MQ2 sensors to monitor temperature, humidity, gas, and smoke levels, transmitting data through MQTT brokers to a cloud-based platform. Machine learning models, including Logistic Regression, Support Vector Machine, Random Forest, and Decision Tree classifiers, are trained and evaluated using real sensor data. Metrics such as specificity, sensitivity, accuracy, F1-score, and area under curve assess the models' effectiveness, while system performance is measured in terms of throughput, latency, and resource utilization. A web-based monitor application visualizes real-time data and alerts users to fire incidents. Experimental results demonstrate the efficiency of integrating WSNs with cloud computing for real-time fire detection and response, which significantly enhances fire management strategies.

1 Introduction

Throughout history, communities have faced various disasters, both natural and human-induced, affecting their homes, work, and commercial environments. Fire hazards impacts and severity are widely recognized. In order to reduce the number of fatalities and property damage, it is necessary to get high-quality fire information as soon as possible and notify the appropriate authorities (Grover et al.; 2019). Early detection and alerting are crucial for evacuation planning and saving lives during disasters. The integration of wireless sensor networks (WSN), cloud computing (CC), and machine learning (ML) has significantly improved fire disaster detection, response, and recovery (Sungheetha and R; 2020).

Traditional fire emergency detection systems have limitations, including being hard-wired and difficult to scale for remote areas, causing slower response times and lack of real-time data processing. High false alarm rates undermine trust and lead to unnecessary evacuations. They are not integrated with other data sources, require manual intervention, and are ineffective in rural or remote areas. Their static nature makes them difficult

to adapt to changing fire conditions. High maintenance costs and regular physical checks further reduce their efficiency (Bal et al.; 2021), (Hu et al.; 2024).

Effective fire emergency management and response measures are needed to minimize the effects of disasters and save lives and property. Big data analytics can improve this process by offering more certain and informed choices, minimizing risk, and increasing operational efficiency. Accurate forecasts of fire incidents enable efficient resource deployment and evacuation, and system optimization is required throughout the management phases due to limited resources. Developing an efficient real-time disaster response system is challenging due to the lack of trustworthy, resilient, and user-friendly platforms and the need for real-time data processing, scalability, and dependability (Sharma et al.; 2020).

1.1 Motivation

According to (Eser; 2024), industrial fires pose a significant threat to the global economy, costing an estimated $37 billion annually. In the United States, 37,000 fires occur annually, resulting in 150 fatalities. Electrical fires account for 20% of all incidents, with manufacturing being the most affected. Welding and cutting activities account for 25% of fires. The food processing industry is particularly prone to fires, with an average of 2,300 incidents per year. Oil and gas facilities have a 40% higher chance of resulting in serious injuries. Construction is the leading cause of fires, with overloaded circuits contributing to 17% of fires. Improper storage of flammable materials accounts for 30% of fires. In 2019, the chemical industry experienced a 12% increase in fire incidents. Faulty equipment is responsible for 28% of fires in mining and 10% in high-rise buildings.

1.2 Research Aims and Objectives

The research goal is to develop and evaluate an efficient cloud-based fire emergency management system that can detect fires in real-time through sensor data processing, simulate fire scenarios, and assess performance based on metrics such as throughput, latency, and resource utilization. To achieve the aim of this study, the following objectives have been defined for this research:

- Develop Fire Detection Mechanism: This process involves designing a detection system capable of identifying different fire sources based on sensor data (e.g., temperature, smoke, gas levels).

- Simulate Fire Events: Use simulation tools (Cooja) to mimic fire scenarios caused by various sources and validate the detection system's performance.

- Implement Real-Time Data Processing: Build a cloud-based data pipeline to ingest, process, and analyze sensor data in real time for early fire detection.

- Evaluation: To evaluate the fire emergency management system, we assess the system based on throughput, latency based on end-to-end delay, computer resource utilization (CPU and Memory utilization).

1.3 Research Question

This study addresses the question: "How effective are cloud-based data processing pipelines in optimizing fire emergency management systems using WSN data?"

In summary, the research aims to create a cloud-based fire emergency management system that can detect fires in real-time using sensor data. The system will identify different fire sources based on sensor data, simulate fire scenarios using simulation tools, and implement real-time data processing through a cloud-based data pipeline. The system will be evaluated based on throughput, latency, and computer resource utilization. The system will also be tested using simulation tools to validate its performance. The ultimate goal is to improve fire safety and efficiency in the future.

This paper is organized as follows. Chapter 1 discusses the background and the context of the research, related work about wireless sensors network, cloud computing and fire detection is presented in Chapter 2. In Chapter 3 the methodology is outlined. The system design and implementation of the fire management system is done in Chapter 4 and the obtained results are discussed in Chapter 5. Finally, Chapter 6 offers a concise conclusion and proposes further research.

2 Related Work

This section presents the recent works reported in the direction prevention of fires in different centers but focusing on the method. The extensive literature survey executed in this article discusses the concept and application of WSN and cloud computing in fire detection and fire detection approaches. By integrating multiple systems into a collaborative network, real-time alerts and enhanced response capabilities can be achieved. This transforms traditional fire safety protocols into modern fire management strategies, improving operational efficiency and safety (Brito et al.; 2020).

2.1 Cloud Computing

For storing large amounts of data, cloud computing provides an affordable and scalable option. According to the National Institute for Standards and Technology, cloud computing enables easy, on-demand network access to a shared pool of reconfigurable computing resources, including servers, networks, and services that can be swiftly provisioned and released with minimal management work or service provider interaction. Cloud computing services may be divided into three groups:

Infrastructure as a Service: This system provides shared basic computer resources, such as networking, storage, processing, and middleware components, through a service. Users may manage load balancers, firewalls, operating systems, RAM, and apps with this computing architecture. IaaS encompasses the hosting, hardware provisioning, and fundamental services required to operate a cloud (Manvi and Shyam; 2013). In essence, IaaS operates on the idea that users "pay for what they need." It gives clients access to high-performance computing. Among the examples of IaaS are Simple Storage Services (S3), Elastic Compute Cloud, and Amazon Web Services (AWS). Online storage is offered by AWS and S3. Clients may readily access the biggest data centers in the world for little fees. Microsoft, HP, and Google are a few IaaS supplier names. IaaS services may be accessed using Google Compute Engine. Additionally, Microsoft offers a cloud platform

with its Windows Azure Platform. Rack Space and NASA collaborated to create HP Cloud (Sandhu; 2022).

Platform as a Service: PaaS is a full-featured cloud-based development and deployment environment that comes with tools to let you build anything from simple cloud apps to sophisticated corporate programs that use the cloud. You use a secure Internet connection to access the resources you need, which are paid for by a cloud service provider. The PaaS service model provides a scalable and adaptable cloud platform for creating, deploying, managing, and running applications (Wulf et al.; 2021). PaaS gives developers what they need to create applications without having to deal with the hassles of maintaining hardware or updating the operating system and development tools. Rather, a cloud-based third-party service provider delivers the whole PaaS environment, or platform (Microsoft; n.d.).

Software as a Service: SaaS eliminates the need for end users to install applications locally. Users can access cloud-hosted apps that are maintained and provided by the provider by using software as a service. SaaS enables online access to and usage of cloud-based applications. Examples of SaaS include calendaring, office 365, Amazon Chime, and email. SaaS is a fundamental cloud computing service paradigm that enables customers to access web-based apps online without having to install them on their PCs (Modisane and Jokonya; 2021). Web browsers are used to access SaaS, which is a collection of services that are developed independently of how they are implemented. It is common practice to employ this approach without realizing its full potential (Ouda and Yas; 2021).

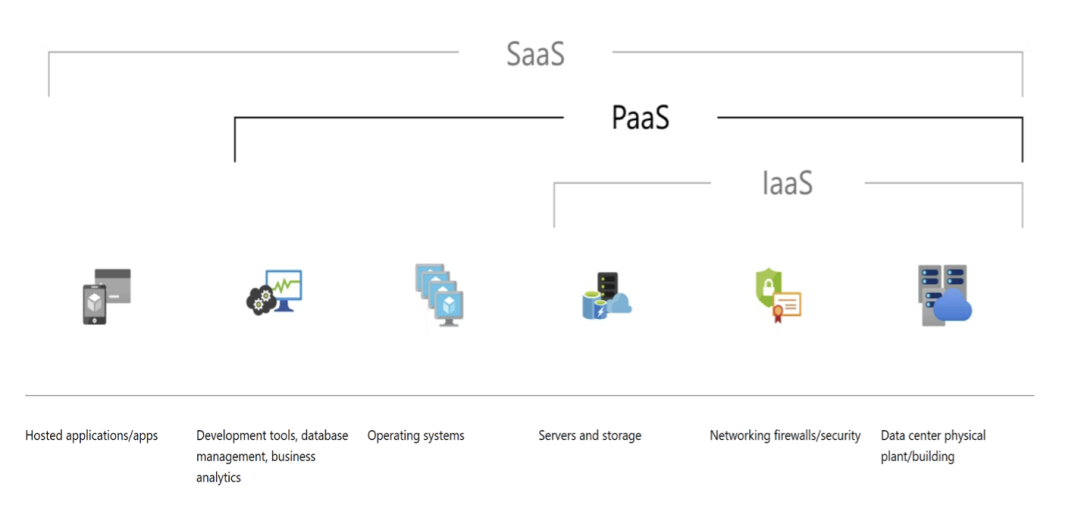

Figure 1: Cloud Computing Architecture (Microsoft Azure, n.d.)

2.2 MQTT

Operating on top of the Transport Control Protocol, the Message Queue Telemetry Transport (MQTT) protocol facilitates communication in an Internet of Things setting. IBM developed the protocol as a lightweight, machine-to-machine communication technique. ISO/IEC 20922 standardized MQTT, and it was also approved as a component of OASIS. Fundamentally, MQTT is a messaging protocol that employs the publish-subscribe communication paradigm. This architecture is ideal for usage in low-bandwidth environments because it eliminates the need for updates from the clients themselves, hence reducing

the amount of resources consumed (Dinculeană and Cheng; 2019). In order to guarantee packet transfer dependability, the MQTT protocol leverages TCP and necessitates a continuous connection between the client and server. As messages must be conveyed as efficiently as possible with little processing power and minimal energy consumption, publish/subscribe protocols like MQTT are said to be becoming increasingly significant, especially for sensor devices in the Internet of Things (Deschambault et al.; 2017). A foundational idea of MQTT architecture is publish-subscribe. The MQTT Broker is used to deliver messages to subscribers when they are published over a particular topic, much like in the broadcasting idea. It is possible to use MQTT over an unstable network. In the event of a connection failure, brokers either guarantee message delivery or employ buffers and push them. This is guaranteed by the architecture chosen for quality of service. To achieve a message in Quality of Services (QoS), MQTT offers three options: Level 1: Send the message based on the network's best effort, at most once ("fire and forget"). At the lowest level of QoS, an acknowledgment is not necessary. Level 2 (QOS2): duplicates may happen, but at least one message can be transmitted. It must be acknowledged. QOS Level 3: A message is delivered using the handshake process precisely once. Although higher QoS levels are more dependable, they also need more bandwidth and delay, which results in higher energy usage (Hamdani and Sbeyti; 2019).

2.3 Wireless Sensor Networks

In several fields, including as business, transportation, the environment, and healthcare, wireless sensor networks, or WSNs, are growing in popularity. Additionally, practically every industry is using Internet of Things (IoT) technology more and more. (Kanoun et al.; 2021) supports this trend. A WSN is a distributed network made up of many small, low-powered, dispersed, self-directed devices known as sensor nodes. Another name for it is motes. Naturally, WSN has limited computer and processing power and consists of a vast number of small, battery-powered, embedded devices that are spread out geographically and are networked to gather, analyze, and communicate data to users. The networks are made up of motes, which are tiny computers that cooperate. However, by merging the many data sources (sensors, databases, etc.), information fusion techniques enable the quality of the reaction to an event of interest to be improved (Varela et al.; 2020). There are now many embedded web server options due to the quick proliferation of IoT devices. Technological developments in industrial wireless monitoring networks (IWSNs) nowadays are mostly focused on applications from supervisory systems, open-loop control systems, regulatory control systems, emergency systems, alerting systems, and monitoring systems (Imran et al.; 2020). Customizing the sensor system with software and hardware technologies would make it simple to deploy for environmental monitoring and offer the option of energy harvesting to power the node in areas with enough ambient radio frequency signals to last for lengthy periods of time (Duobiene et al.; 2022).

(Bouabdellah et al.; 2013) opined those traditional methods were ineffective because human observation towers are unreliable and challenging living conditions, and satellite-based forest fire detection systems like MODIS and AVHRR, are limited by terrain, time of day, and weather conditions. Recently, WSN technology has emerged and been adopted by several countries. WSNs must consider design goals such as energy efficiency, early detection, accurate localization, forecast capability, and adaptability to harsh environments. Research on forest fires using WSNs has been conducted worldwide, and other interesting investigations have also been conducted in this area. However, due to its

unreliable topology, scarcity of resources, and lack of infrastructure, WSNs encounter a number of difficulties. Maintenance is challenging since nodes are dispersed upon deployment. They only use non-rechargeable batteries, which limits their capacity for memory, compute, and power. With problems like conflicts, latency, and node synchronization, communication and data transfer are frequently unstable. System performance and dependability are further complicated by handling node failures, changing the topology, and adding or removing nodes. Another crucial issue is designing sturdy hardware for sensor nodes (Sharma et al.; 2013).

2.4 Machine Learning

The general process of learning involves changing or gaining new behaviors, values, knowledge, abilities, or preferences. The theory of personal learning, or how people learn, is defined by behaviorism, cognitivism, constructivism, experiential learning, and social learning. In contrast to people, who naturally learn from experience, machines rely on data. Fundamentally, machine learning (ML) is a branch of artificial intelligence that gives computers the ability to think and learn independently. Large volumes of training data are frequently needed by machine learning algorithms in order to create an appropriate model. Thus, gathering a sizable collection of representative training examples and storing them in a format appropriate for computation is a crucial initial step in using machine learning techniques (Alzubi et al.; 2018). Essential to fire detection systems, machine learning (ML) models use training data to translate input variables (like sensor readings) to outputs (like fire or no fire), allowing for precise forecasting and quick reactions to possible fire situations. Through data processing and performance criterion maximization, machine learning offers methods that may automatically construct a computer model of these intricate interactions. Training is the automated process of creating a model, while training data is the information needed to train the model. It is possible to utilize the trained model to predict new input values that were not included in the training data and to get fresh insights into the mapping of input variables to the output (Başţanlar and Özuysal; 2013).

2.5 Fire Detection

To prevent fire-related damage to cars, (Sowah and et al.; 2016) presented a system for detecting car fires that uses an Arduino microcontroller and fuzzy logic, an artificial intelligence technique. When a fire is detected, temperature, flame, and smoke sensors are utilized to raise an alert and provide carbon dioxide to the area. On a medium-sized physical vehicle, a system is installed and tested, and a 2 kg cylinder is placed behind the passenger seats.

To keep an eye on environmental factors like temperature, humidity, and smoke, (Al-Dahoud and et al.; 2023) proposed a WSN and IoT-based, low-cost forest fire warning system made up of a network of strategically positioned sensors across the forest. Principal component analysis and wavelet packet decomposition were utilized to identify and forecast the incidence of forest fires using the sensor data that was sent to a central computer. Through a mobile application that displays the fire maps and the most recent updates, the system sends out real-time notifications to users and forest authorities. Experiments have been used to analyze the suggested system, and the findings indicate that it may efficiently detect forest fires with low cost, low false alarms, and high accuracy.

(Saputra et al.; 2017) created a wireless Internet of Things (IoT) fire alarm system that use an ad hoc network with several nodes positioned thoughtfully across the house. A nodeMCU and sensors that use the SigFox LPWAN protocol to detect smoke, humidity, temperature, flame, carbon monoxide, and methane are installed in each of these nodes. Every node establishes its own wireless network. A Raspberry Pi microcontroller and a 4G module are used by the central node to connect with the other nodes. When a fire is discovered, the node alerts the fire department, calls the user, activates the local alarm system via SMS, and transmits a signal to a centralized node. This tactic won't be as effective for you if you are unable to access the internet or if your connection is bad.

(Kizilkaya and et al.; 2022) introduced a new hierarchical method for detecting forest fires. The suggested approach minimized the transmission of visual data by combining scalar sensors and multimedia in an organizational structure. To increase detection accuracy and decrease traffic between the edge devices and the main station, a lightweight CNN model was created for devices at the network's edge. A testbed, network simulations, and 10-fold cross-validation were used to assess the framework's energy efficiency and detection accuracy.

(Hamzah and Abdul-Rahaim; 2022) explored low-cost industrial and household safety systems by integrating innovative design methodologies and WSN. Smoke, humidity, and temperature sensors, were employed to monitor the environmental parameters and ensure precise system responses. Transmission of sensor data was facilitated using a cloud gateway to enable real-time monitoring. The system was designed to be accessible for general use, while notifications are used to alert users. Sensor data, such as gas, flame, temperature, and humidity readings, are sent to smartphones, and when thresholds are exceeded, the system provides fire coordinates and sent alerts to mobile devices.

(El-Hosseini and et al.; 2020) developed a power-efficient IoT-based fire detection model for smart cities, using multi-functional sensors and a partial coverage approach. The model uses sleep scheduling to conserve sensor energy and integrates fog computing for real-time data processing. Validated through simulation and experimental implementation, the model optimizes sensor usage and extends network lifetime. Compared to current methods, it reduces active nodes by 64.33% and 15% and increases network lifetime by 91.32% and 12%, demonstrating superior energy efficiency and network longevity compared to MWSAC and PCLA, respectively.

Centralized control and building automation are the two areas where wireless communication technology is most commonly used. (Yunhong and Meini; 2016) suggested software with a wireless sensor fire detection system activated. The technology helps in the evacuation process by monitoring the fire alarm from a great distance; nevertheless, the control center's slow response time affects the response time. Its inability to connect to all sensors via the main server is the cause of this.

A small, inexpensive device for detecting fires is the Fire Monitoring and Warning System (FMWS). (Sarwar and et al.; 2018) proposed starting with a flame sensor for detecting a fire, it employs a step-by-step methodology. If identified, the system uses rule-based logic to evaluate temperature and humidity data in order to determine the severity of the fire. While medium-intensity flames set off water showers and sirens, low-intensity fires set off warnings, and high-intensity fires start more intense reactions. For simulations using MATLAB, the data is sent to the microcontroller, where it is examined and recorded in real time. In order to guide response mechanisms, FMWS responses are simulated using fuzzy logic.

(Karaduman and et al.; 2018) proposed an FDS that increases throughput and covers a

large physical area by utilizing a multi-hop wireless sensor network and a library based on ContikiOS. The system also has a clouding feature that stores data online using Google's Firebase and an Android app to notify users in case of an emergency. Furthermore, the network may expand without needing more sink nodes to gather data. One benefit of a multi-hop network is that, in the case of limited transmission power from the sink node, extra source nodes can provide temperature values through relay nodes. The results of the test demonstrate that the system responded to temperature changes using a multi-hop network and produced a warning or alert message in around 4 seconds. The technology may also alert users in the event of an emergency.

Studies on the combination of cloud technologies with WSNs have been conducted. A service is used to fulfill user requests in cloud computing, according to a study (Ahmed and Gregory; 2011). Additionally, a data center receives gathered data on a regular basis. Technology from cloud computing is employed to do this. It may be difficult to integrate cloud computing and WSN technologies. Integration between cloud computing and WSN is still being developed. New methods for integrating these two technologies are presented in certain works, such as (Dwivedi and Kumar; 2018), which also include analytical and experimental findings.

Energy efficiency is not included in certain research since the proper evaluation techniques are not used. Some simulation techniques do not take multimedia data into account when evaluating detection accuracy, even when they demonstrate the suggested designs' energy efficiency. The categorization process is not carried out at the network edge by systems that employ deep learning or machine learning models. This study discusses a development in fire detection techniques that uses new sensors and methods to identify fires more quickly.

2.6 Chapter Summary

This chapter reviews advancements in fire prevention systems, with a focus on WSNs, cloud computing, and ML. Cloud computing provides scalable data storage and processing infrastructures, platforms and services that are cost-effective through services like Amazon Web Services. It supports extensive fire detection data analysis via IaaS, PaaS, and SaaS models. MQTT is a IoT communication protocol that is lightweight, uses low-bandwidth with a reliable publish-subscribe model; and because of the resource constraints of IoT sensors, it is ideal for real-time fire detection. WSNs enable early fire detection and real-time monitoring using distributed sensor nodes. Despite challenges like limited battery life and maintenance, innovations such as energy harvesting and robust hardware enhance their functionality. Machine learning (ML) improves fire detection accuracy by processing sensor data to identify patterns based on training data to predict and respond to fires effectively. Integrating cloud computing, WSNs, and ML enhances fire detection accuracy, reduces response times, while improving management strategies. Addressing challenges like energy efficiency and network reliability remains crucial for broader adoption.

3 Methodology

In this section, we discuss the methodology used to build the FERS architecture designed to facilitate data acquisition, transmission, and real-time monitoring and analysis through cloud services. The WSN node is equipped with sensors for detecting environmental

anomalies, enabling prompt detection and response. Data from each sensor is processed locally before being transmitted via a series of cloud and streaming services for more advanced analysis.

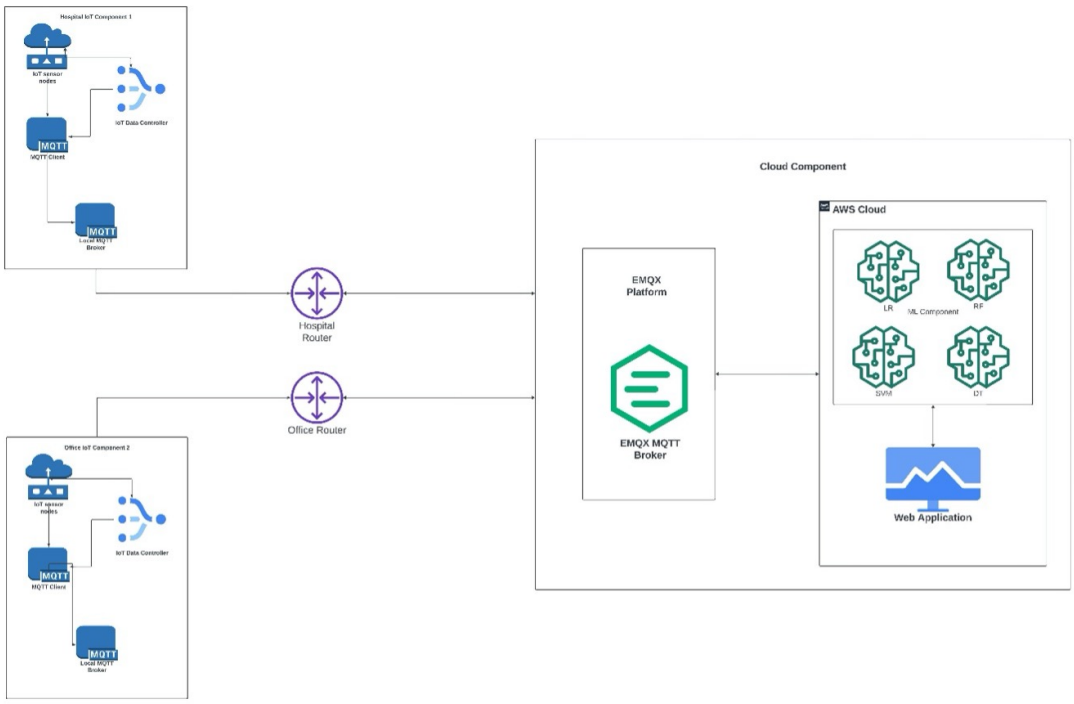

Figure 2: Architectural Diagram for Proposed Fire Detection System

The FERS integrates Wireless Sensor Networks (WSNs) with cloud-based platforms to ensure efficient monitoring, real-time data processing, storage, and machine learning (ML) analysis. The architecture consists of three main modules: the IoT-enabled system, the machine learning module, and the monitoring application. Each module works in tandem to provide a scalable, dependable, and real-time fire detection and response solution. To build the FERS to be scalable, dependable, and to be able to detect fire in real time, the FERS leverages IoT devices and a cloud system. The architecture combines cloud infrastructure, machine learning models, IoT sensors, MQTT brokers, and a monitoring web application. The IoT sensors used gathers environmental data such as temperature, humidity, and LPG, smoke and carbon monoxide. These sensors use a fast, lightweight MQTT protocol to send data to the Mosquito MQTT broker when they are part of a wireless sensor network (WSN).

The Mosquitto broker acts as a middleman that controls the data flow locally, through data collation and forwards it to the cloud-based EMQX broker. By connecting local and remote topics via an MQTT bridge (or the local to the cloud), the EMQX broker facilitates seamless communication and makes sure that data transmission is done in real-time with high-throughput data processing. In order to control the data flow, the IoT Data Controller simulates different signals, such as fire or normal, and makes sure that they are regularly sent to the MQTT client for additional analysis. Machine learning algorithms examine incoming data from the EMQX broker to identify abnormalities that predicts if there is a fire or not. These models are trained using datasets gathered from the sensors. These analysis' (whether there is fire or not) are shown on the monitor web

application, which is hosted on an AWS EC2 instance. The application gives users real time update as to the state of each sensor node by updating the environmental status in real time.

3.1 The IoT Component

The IoT component system is primarily responsible for handling the data that is initially collected from the sensors. This component is made up of the IoT sensor nodes, the IoT data controller, the MQTT client, and the local MQTT broker

3.1.1 IoT Sensor Nodes

The IoT sensor nodes includes a suite of sensors for monitoring key environmental parameters associated with potential fire risks or events. The DHT11 and MQ2 sensors are implemented for early and reliable detection. The DHT11 sensor device is a digital sensor device that commonly used to measure temperature and humidity. It is ideal because of its low power usage (0.3 mA in standby and 60 mA when active), and enables long-term monitoring in remote settings. It has an accuracy of ± 1°C for temperature and $\pm 1\%$ for humidity, respectively. It is designed to be accurate, efficient, compact and adaptable. Additionally, the MQ2 is a multipurpose gas and smoke sensor that is frequently used to detect carbon monoxide (CO), LPG, and smoke amongst other gases. When the gas comes into contact with the sensing material, the resistance of the material changes, which is the basis for gas detection. It is possible to monitor gas concentrations using a basic voltage divider network. Its wide detection range, low cost, long lifespan, high sensitivity, and fast reaction time make it the perfect sensor. The MQ-2 is mostly utilized in fire alarm systems for smoke detection, but it may also be used for air quality monitoring and gas leak detection.

3.1.2 IoT Data Controller

The IoT data controller is a critical part of the IoT component. The data controller manages how the data is simulated i.e how the data flows from the IoT sensor nodes to the MQTT client. The controller sends the type of signal needed to test the FERS. It sends a normal or fire signal or data to the MQTT client which is then sent to the cloud. The signal remains the same until it is changed by the IoT data controller.

3.1.3 Mosquitto (MQTT Broker)

The Mosquitto broker acts as the central hub for MQTT communication. Messages are exchanged between clients (devices) by how it handles managing topics. Publishers transmits the messages to certain topics in this system, and subscribers listen to these topics to get the messages that are pertinent to them. The broker manages features like reliably delivering messages, supports Quality of Service (QoS) at various levels, and handles retained messages and persistent sessions. It is very scalable and lightweight which makes it ideal for systems with constrained resources and large-scale IoT deployments. To further guarantee privacy and integrity, the broker also offers secure communication with TLS and authentication methods.

3.1.4 Mosquitto (MQTT Local Client)

To publish and subscribe to messages, the Mosquitto clients uses mosquitto$_pubandmosquitto_subcomman$
$linetools, respectively. In order to test and debug MQTT systems, these clients provide basic examples of M$
$timepublisheddata. These tools demonstrate the decoupled nature of MQTT communication, in which subs$

The Mosquito broker and clients work together to create a strong communication structure within the IoT component and the cloud component via an MQTT bridge. Several clients might post and subscribe to different topics since the broker acts as a middleman. For instance, a sensor may publish temperature data to a topic called sensor/temperature, to which several devices, including alarm systems or dashboards for monitoring, may subscribe in order to get updates. This design supports a variety of application situations by enabling scalability and flexibility.

3.2 Machine Learning Module

Machine learning offers a predictive model that allows for early detection, high accuracy and real-time processing of data which may help with predicting and detecting fires in real time. They are capable of identifying patterns and anomalies in sensor data, and process these data in real-time; this is beneficial in reducing false alarm rates, detect fires in their early stages enabling rapid response to potential fires and minimizing the time it takes to detect and extinguish them.

The bridge between the EMQX and the Mosquito MQTT brokers makes data transmission from the local IoT component to the cloud seamless. MQTT connects local topics to EMQX after Mosquitto manages local sensor data and publishes it to local topics. Cloud services like the monitoring web application and machine learning module are integrated with EMQX to enable remote data transfer. Once trained, the machine learning models analyze streams of incoming EMQX data in real time, searching for anomalies or trends that may indicate a fire. For accurate fire detection and response, this bridge ensures reliable, efficient data exchange.

In the machine learning module, four models were implemented: logistic regression (LR), support vector machine (SVM), decision tree (DT) and random forest (RF). A brief description of these models are presented below:

Logistic Regression (LR): Logistic regression is a statistical technique used for binary classification tasks, such as identifying whether an environment is in a normal state or experiencing a fire incident. It uses a logistic function to describe the link between input sensor measurements (such as temperature, smoke levels, and gas concentrations) and the likelihood of a fire. Labeled sensor data is used to train the model, and the cross-entropy loss is minimized to estimate the parameters. After being trained, LR can use projected probabilities to identify fresh sensor data, which makes it appropriate for real-time fire detection.

Random Forest (RF):

Random forest is a versatile machine learning algorithm used for classification tasks, including fire detection. A randomly chosen subset of sensor data and characteristics is used to train each of the many decision trees it builds. All trees analyze the sensor data during categorization, and the ultimate result is decided by the majority vote. RFC is very good at properly recognizing fire occurrences because it can handle complicated interactions between sensor readings, huge feature spaces, and nonlinear connections.

Support Vector Machine (SVM):

SVM is a robust algorithm used for fire detection by classifying data into normal or fire-incident categories. It finds the best hyperplane to divide sensor data into different classes while optimizing the distance between the hyperplane and the nearest data points. This makes it possible to recognize fire events from sensor inputs with great accuracy and generalization.

Decision Tree (DT):

Decision tree method is a straightforward yet powerful tool for detecting fires. With little assistance from the user during training, it divides sensor data according to feature thresholds and utilizes a tree-like structure to categorize the data. The intuitive nature of DT and its capacity to capture nonlinear feature connections make it very useful for recognizing fire situations.

These machine learning models will be trained and evaluated using the dataset. The machine learning module analyses the incoming stream of data from the EMQX broker. The analyses is done after the machine learning models have been trained using the dataset.

Upon entering the AWS environment, data undergoes several processes. For the data to be used to train the machine learning models, they are first preprocessed. This means that the data is checked to find any missing values. If there are any missing values, they are removed from the dataset and the dataset is cleaned. After the dataset has been cleaned, it then split into training and testing datasets that will be used to train and evaluate the trained models' performance.

3.3 The Monitor Web Application

The EMQX broker organizes data as topics, which are subscribed to by the web application hosted on AWS. The application uses WebSocket protocols to periodically request and receive updates from the EMQX cloud MQTT broker. Once received, the web application processes the data and dynamically updates its user interface, displaying the current system status (e.g., normal operation, fire alert, or system off). This connection ensures real-time data visualization, enabling users to monitor the environment effectively and respond promptly to alerts.

This methodology ensures efficient, real-time processing and analysis of WSN data, using a scalable and resilient cloud-based architecture to support complex monitoring and predictive needs.

3.4 Evaluation

The system's efficiency is evaluated in two parts i.e the machine learning models' performance and the proposed system's performance. The ML models were evaluated based on their accuracy, F1-score, area under curve, sensitivity and specificity. The proposed system's performance were assessed based on their throughput and latency (end-to-end delay).

3.4.1 Machine Learning Model Evaluation

Accuracy: The accuracy of a machine learning model is expressed as a percentage of all correctly predicted outcomes. By dividing the number of examples that were properly categorized by the overall number of instances in the dataset.

Area Under Curve (AUC): The area under the Receiver Operating Characteristic (ROC) curve, which is a plot of true positive rate (TPR) against false positive rate (FPR) at various categorization thresholds, is represented by the AUC. The TPR denotes the proportion of true positive cases that are properly predicted as positive, whereas the FPR denotes the proportion of true negative instances that are wrongly projected as positive. F1-Score: The F1-score is used to assess a model's general accuracy. It is the harmonic mean of recall and accuracy; two additional crucial metrics used to assess the performance of the models. The degree to which positive forecasts turn out to be real positives is known as precision. The percentage of good outcomes that were accurately projected as positive outcomes is known as recall.

Sensitivity: Sensitivity, also referred to as the true positive rate, quantifies how well a model can detect positive events among all real positive instances. The percentage of true positives—or positively anticipated outcomes—to the total of true positives and false negatives is how it is computed. A high sensitivity suggests that the model successfully catches the majority of positive cases, reducing the false negative rate and the number of real positives that the model misses.

Specificity: The capacity of a model to accurately identify negative occurrences out of all the real negative examples in a dataset is measured by a parameter called specificity. Another name for it is the genuine negative rate. Put another way, specificity is the degree to which a model can precisely forecast the lack of a condition or class—that is, the negative class—that it is attempting to uncover. With a high specificity, the model can effectively reduce false positives and provide an accurate prediction about whether a case actually falls into the negative group.

3.4.2 Proposed System Evaluation

Throughput: Throughput is the actual rate at which data is successfully transmitted over a network or channel within a specific period. Also known as the effective data rate or payload rate, it reflects the real-world performance of the network. It is useful for assessing the actual performance and efficiency of the network.

Latency: In a network, during the process of data communication, latency(also known as delay) is defined as the total time taken for a complete message to arrive at the destination, starting with the time when the first bit of the message is sent out from the source and ending with the time when the last bit of the message is delivered at the destination.

3.5 Deployment

IoT-enabled sensors, a MQTT-based data pipeline, and AWS-hosted machine learning analysis are all integrated into the deployed system. Data is sent to the cloud via a bridge between the local Mosquitto MQTT broker and a public EMQX MQTT broker. Continuous monitoring and fire event response are ensured via a Python-based monitoring application that displays real-time sensor data and ML analysis outputs on a web platform hosted on AWS.

This methodology ensures efficient, real-time processing and analysis of WSN data, using a scalable and resilient cloud-based architecture to support complex monitoring and predictive needs.

4 Design Specification

The FERS that we propose is aimed at enhancing fire detection and response efficiency by integrating IoT, machine learning and cloud-based processing. The primary objective of the proposed system is to enable real-time detection of fire incidents, give timely alerts, and to provide dynamic monitoring through a web-based application. The system is designed to be scalable and reliable, the system addresses the need for continuous sensor data gathering, quick data analysis, and effective visualization for decision-making. This section outlines the functional and technical requirements along with the system's architectural design, ensuring a comprehensive understanding of the system's development and capabilities.

4.1 Functional Requirements

There are several key functions to that the proposed system must perform in order to deliver a robust and effective fire detection solution. First and foremost, the system should continuously collect data (temperature, smoke, and gas readings) from the IoT sensor nodes and and transmit it for processing. The data collected from the sensors must be transmitted efficiently using the MQTT protocol to make sure that latency is kept low and guarantee delivery even in unstable network conditions. Lastly, one core functionality of the system is for it to have the ability to analyze and classify the data in real-time using a machine learning-based classifier that distinguishes between normal and fire-related conditions. Also, the system must feature an application for real-time monitoring by visualizing the status of the sensors and fire alerts, which will offer an intuitive and responsive user experience. It should issue notifications immediately upon detecting fire signals, making sure that awareness to incidents are timely. Finally, scalability is a critical functional requirement that will enable the system to accommodate increasing number of IoT-enabled sensors and the data that they will generate without compromising performance or reliability.

4.2 Technical Requirements

The technical foundation of FERS includes specific hardware and software resources to achieve its operational objectives. On the hardware aspect, IoT-enabled sensors capable of detecting key environmental elements such as temperature, gas and smoke are essential. The system needs a connection to the internet via a router. The system also requires a local computing environment to run the Mosquitto MQTT broker and the Contiki-NG simulator for emulating sensor behavior. Cloud resources, particularly an AWS EC2 instance, support data scalability and availability.

Software requirements are centered on programming languages and frameworks optimized for real-time processing and machine learning. Python is used extensively for backend development, machine learning model integration, and data analysis. The FastAPI framework facilitates the development of robust APIs, while Paho MQTT manages the messaging. Scikit-learn and Pandas provide tools for training the models, evaluation, and analyses of the data. The HTML, CSS, and JavaScript to create a dynamic user interface. The system's architecture harnesses a modular design that enables the independent IoT components for data acquisition, machine learning classification, and monitoring. MQTT serves as the messaging protocol, linking local and cloud environments via a

bridge configuration for seamless data flow.

4.3 System Design

The system design for FERS is structured into three interrelated components that collectively deliver a comprehensive fire detection and response solution.

The IoT-enabled system creates the bedrock for collecting and transmitting data. The sensor data is generated through simulation by using the Contiki-NG operating system, it generates values at set intervals and publishes them to a local Mosquitto MQTT broker under a designated topic. This configuration ensures reliable data exchange within the local environment. To extend the system's scalability, a bridge connects the local broker to a cloud-based EMQX broker, facilitating the transmission of sensor data to remote systems for processing and storage.

The machine learning component provides the core analytical capabilities of FERS. Four distinct machine learning models—Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine—are trained using preprocessed sensor data that includes both normal and fire-related events. Preprocessing steps involve data normalization, label encoding, and handling missing values. The models are optimized through hyperparameter tuning and evaluated using metrics such as accuracy, precision, recall, and F1-score. A classification pipeline automates data normalization and encoding while predicting sensor data classifications in real-time, triggering appropriate fire or normal state responses.

The monitoring application is used for visualizing sensor activity and fire events provided by a user-friendly interface. The frontend which is built using HTML, CSS, and JavaScript, supports dynamic data updates, using WebSocket to retrieve and display new messages every two seconds. The backend which is powered by the FastAPI, processes incoming data streams, integrates machine learning predictions, and serves the API endpoints for frontend interaction. The design ensures that instant feedback on system status is sent to the users via the user interface, and that it can simulate fire scenarios using the control buttons, enhancing both usability and testing flexibility.

5 Implementation

In this chapter, we present the steps taken to meticulously implement the proposed system. This chapter includes the processes, methods and design parameters implemented to execute the FERS. It discusses in great detail, how the IoT components were chosen, and configured. It describes how the machine learning module was built, trained and evaluated. It extensively describes how the monitor web application was designed. Finally, we describe how all this components were stacked together to deliver the final proposed system.

5.1 System Implementation

To implement the fire detection system, three system components were created, namely – IoT-enabled system, the machine learning component and the application monitor.

5.1.1 IoT-Enabled System

The design and implementation of an IoT-enabled system was done to maximize the scalability and speed of cloud-based data processing pipelines for Fire Emergency Response Systems (FERS). The system incorporates an MQTT client application running in the Contiki-NG OS simulator, a local Mosquitto MQTT broker, a local Mosquitto MQTT client, and an IoT Data Controller application. Real time data processing and analysis are made possible by the effective data flow that these elements enable from nearby sensors to the cloud. The MQTT Mosquitto broker, which serves as a data exchange middleman, is at the heart of the local IoT configuration. The broker was set up to receive and organize data from the local environment primarily through the subscription to a specific topic - Local/FERS/IoT-Data/json. In order to supplement this, the MQTT client application was created and run within the Contiki-NG OS simulator. It was designed to read sensor data and broadcast it on a regular 5 second interval. This was done to balance the publishing timeliness and system resource usage. Sensor data was sent to the Local/FERS/IoT-Data/json topic, where the local MQTT broker could access it. The local Mosquitto MQTT client is configured to subscribe to the local topic - Local/FERS/IoT-Data/json on the local broker. In order to make sure that the sensor readings are sent correctly to the pipeline's downstream components, this client verifies and tracks data broadcast by the MQTT client application. The Contiki-NG OS simulator's implementation of the MQTT client application mimics the system's IoT devices. This application receives sensor data and publishes it to the topic Local/FERS/IoT-Data/json on a 5-second interval. The data generated by the simulated sensors is used to mimic the FERS system's reaction capabilities in relation to fires. The MQTT client application has the ability to publish certain types of sensor data, because it is dynamically controlled by the IoT Data Controller application. 10% of the original dataset was set aside so that it could be used by the application. The reserved dataset is divided into two equal parts between sensor signals for fire and normal conditions. The user interface of the IoT Data Controller application shows the sensor data that is being streamed at the moment. It has two buttons mapped to fire and normal data type which gives users control over the signals that are published, because they can easily switch between these classes in real time. This feature is crucial for testing the system's performance and recreating various scenarios. A bridge was then set up between the local Mosquitto MQTT broker and a public EMQX MQTT broker to create a link to the cloud component. The bridge connects the remote topic Remote/FERS/IoT-Data/json on the EMQX broker to the local subject Local/FERS/IoT-Data/json so that data can be transmitted to the cloud, ensuring that the system is scalable and reliable.

5.1.2 Machine Learning Component

A key element of this research is the machine learning component, which is intended to evaluate and identify fire signals in real time for the Fire Emergency Response System (FERS). Four machine learning models are included in this component namely - logistic regression, decision tree, random forest and support vector machine. These models were meticulously trained, assessed, and put into use to effectively identify and categorize fire signals from IoT sensor data in real time. The implementation began with the preparation and preprocessing of a historic sensor dataset. This dataset contained labeled sensor readings, including normal and fire-related signals, which were essential for training the models. Additionally, the dataset needed to be preprocessed so that the machine learning

models could use it for analysis. These included label encoding which converts category variables to numerical variables. Also, sensor readings were standardized using normalization, and missing values were handled through data cleaning. After preprocessing, 20% testing dataset and an 80% training dataset were created from the remaining 90% dataset. This guaranteed the models' ability to learn well and be evaluated objectively. The LR, SVM, RF, and DT machine learning models were trained to accurately identify fire signals using the training dataset. The prediction ability of each model were improved by optimizing its hyperparameters according to the features of the dataset. After that, the trained models were used to predict the labels of unseen data in order to assess them using the testing dataset. Confusion matrices for each model were created by comparing these predictions with the actual labels. This provided a thorough analysis of true positives, true negatives, false positives, and false negatives.

To quantitatively assess the performance of each model, five key evaluation metrics were computed: accuracy, sensitivity (recall), specificity, area under the curve (AUC), and F1-score. These metrics provided a comprehensive evaluation of each model's strengths and weaknesses, enabling an informed selection of the most suitable model for deployment. After evaluation, all trained models were saved as serialized objects, ready for integration into the FERS pipeline.

The training procedure was created such that it is a reusable class solution for processing IoT sensor data in real time. This comprised of a simplified classification workflow, model integration and automated data preprocessing. Incoming sensor data is processed in real time by the pipeline, which also normalizes and encodes it, classifies the data using the learned models, and instantly indicates if the signals point to a possible fire or normal activity.

The response system and data streams work seamlessly together because of the ML component's real-time interaction with the IoT component. The connection makes it possible for the FERS pipeline to detect fire signals in real time, process sensor data continually and also scale effectively to accommodate growing data volume.

5.1.3 The Monitor Application

Successful identification of fire events and provision of real-time information on the behavior of IoT-enabled sensors is dependent on the crucial part played by the monitoring web application of the FERS. Utilizing the cloud architecture provided by the AWS EC2 instance ensures scalability and high availability. HTML, CSS, JavaScript, and Python were used to create a dynamic and adaptable frontend for monitoring and visualization to seamlessly integrate backend statistics.

Backend Architecture and Implementation

The backend of the monitoring application was implemented using Python and its libraries such as FastAPI, Paho MQTT, Scikit-learn, and Pandas. FastAPI is the core of the program and is known for its excellent performance and ease-of-use. It makes server-client connection and API creation easier. This handles the data streams and carriesout analytics in real time, and giving the application a solid base. Key backend functionalities include API endpoints, WebSocket integration, and MQTT-based data handling. FastAPI provides endpoints to manage the home page, access data streams, and transmit analyzed IoT sensor data to the frontend. A WebSocket interface enables real-time updates, ensuring the application's state reflects the latest data dynamically. The Paho MQTT library configures a client that listens to the Remote/FERS/IoT-Data/json

topic on the EMQX MQTT broker. The MQTT client connects to the broker on port "1883" with a keep-alive interval of "60" seconds. Incoming messages are processed asynchronously, analyzed using trained machine learning models ,and queued for frontend visualization. To assess system performance, the latency and throughput metrics were calculated. Latency was measured in the on_message function by taking the recorded timestamps at the start and end of message processing to obtain the processing time. The average latency was calculated by dividing the sum of all the latencies by the total number of processed messages, while the throughput was calculated by tracking the number of messages published over the elapsed time. This provided insights into how the system was efficient in handling data streams. The backend was packaged as an Asynchronous Server Gateway Interface (ASGI) which enables high concurrency and seamless integration with real-time WebSocket-based updates.

Frontend Architecture and Visualization

HTML, CSS, and JavaScript were used to develop the application's front end. The design was made to to have a user-friendly interface for real-time monitoring and visualization. The application was structured and styled using HTML and CSS to provide an aesthetically pleasing layout. We implemented real-time updates using JavaScript. The frontend establishes a WebSocket connection with the backend so that the queued messages can be retrieved every 2 seconds. The interface then displays the notifications, which include the present state of IoT devices and any fire events that have been detected. This dynamic technique improves situational awareness and decision-making ability by making sure that the users receive instant feedback. In summary, this system demonstrates a robust and scalable approach to integrating IoT technologies with machine learning and cloud-based data processing pipelines, providing an essential foundation for efficient and timely fire emergency response operations.

6 Evaluation

In this chapter, the outcome of this research study is presented. The machine learning models and the proposed system is evaluated and discussed.

6.1 Evaluation of Machine Learning Module

For real-time fire detection, this study's Fire Emergency Response System (FERS) was created by combining cloud-based, user-friendly monitoring software, IoT sensors, and machine learning algorithms. The temperature, humidity, LPG, CO and smoke levels were reliably monitored by the DHT11 and MQ2 sensors, which were the environmental parameters measured for fire detection. The machine learning models - logistic regression, support vector machine, random forest, and decision tree were trained and tested using a dataset containing the IoT sensor data readings. The models were evaluated based on accuracy, F1-score, area under curve, sensitivity, and specificity. Metrics like accuracy, F1-score, AUC, sensitivity, and specificity are crucial for evaluating how well machine learning models perform, particularly in situations where accuracy and dependability are crucial, like fire detection. A thorough grasp of the model's performance may be obtained by analyzing these metrics. This guarantees a balance in fire detection between detecting fires (sensitivity), preventing false alarms (specificity), and preserving overall robustness and dependability (F1-score, AUC, and accuracy). This comprehensive assessment facilitates the implementation of reliable, efficient, and actionable systems. When

the outcome of the models' were obtained, the models demonstrated exceptional ability to differentiate between a normal signal and fire signal. The models all showed 100% across all the performance metrics.

To examine why the machine learning models were so accurate and obtained perfect score across all the metrics, we observed the values in the dataset. We observed that for the dataset where the fire signals were true, the LPG, CO and smoke values were highly distinguishable from the dataset where the fire signals were false. Because of this observation, it was concluded that the classes were palpably easier for the machine learning models to identify and predict when there was a fire occurrence. The models' real-time analysis of incoming sensor data and its ability to accurately predict fire occurrence engendered it's proactive detection and response, ensuring accurate and timely fire alerts. Quantifying how well the proposed system performs solely based on the machine learning's performance was insufficient. Thus, the network performance was assessed based off of the latency and the throughput. Low latency and high throughput work together to create a balanced and effective network that can manage large data loads without experiencing any delays and ensures fast data delivery. Because they have a direct impact on the effectiveness and dependability of the system, latency and throughput are crucial measures for evaluating network performance, especially in systems like IoT-enabled fire emergency response systems. A balanced and effective network is produced when low latency and high throughput work together to provide fast data transfer and the capacity to manage large data loads without interruption. These indicators assist in locating bottlenecks, allocating resources as efficiently as possible, and confirming that the network is prepared to accommodate FERS and other mission-critical applications.

6.2 Evaluation of FERS System

When the latency and throughput were calculated, it was observed that in a 1 minute period the range for latency was between 0.0249 and 0.0667s, while the throughput range was between 0.07 and 0.14 messages/second. In the same 1 minute period, the average latency and throughput was calculated. It was observed that the average for the latency and throughput were 0.0602(s) and 0.13 messages/second, respectively. It is worth highlighting that these values were obtained based on the calculations from 2 IoT sensor nodes.

To conclude the evaluation, the monitor application, which was installed on AWS, had an intuitive interface for visualizing system conditions and providing useful feedback. Three main states were demonstrated by its outputs: Normal Operation shows which devices indicated consistent readings with a "Normal Signal Detected" status depicted with a green highlight colour as shown in figure 4 below. Fire Alert, which was activated by anomalous readings like high levels of smoke, CO, and temperature, indicating a "Fire Signal Detected" was illustrated with a red colour as shown in figure 4 below. And the System Off showed when the monitoring system was turned off and not in use as shown in figure 3 below. The integration of cloud computing, machine learning, and IoT technologies produced a scalable and dependable fire detection system, fulfilling the goals of the study and proving its usefulness in real-world applications.

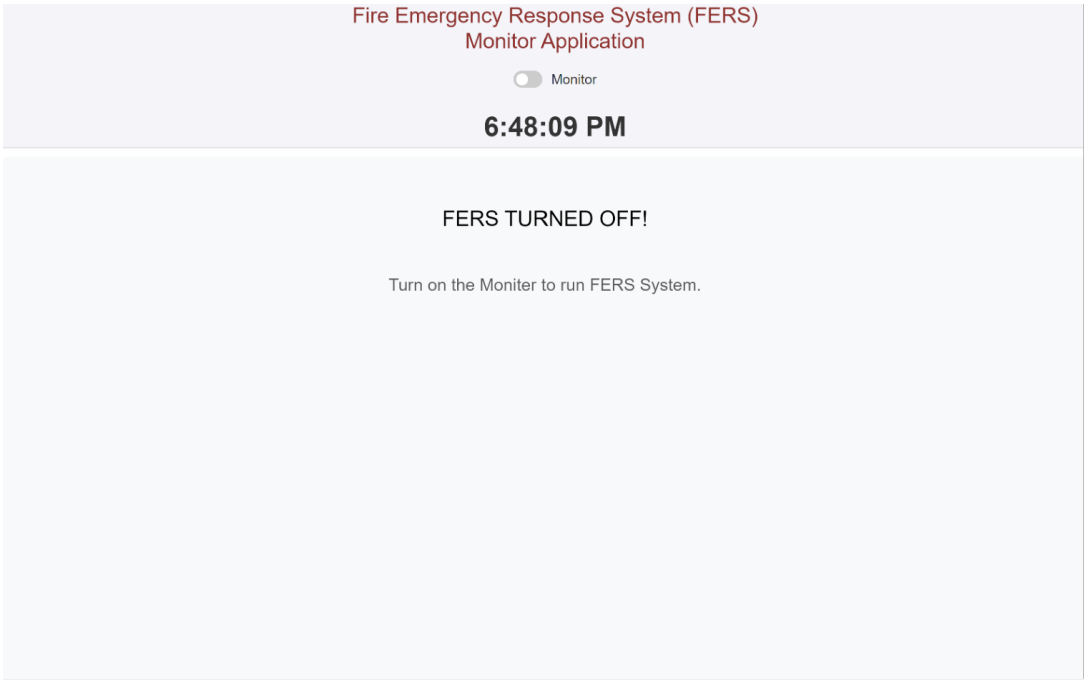

Figure 3: FERS Monitoring System when Off

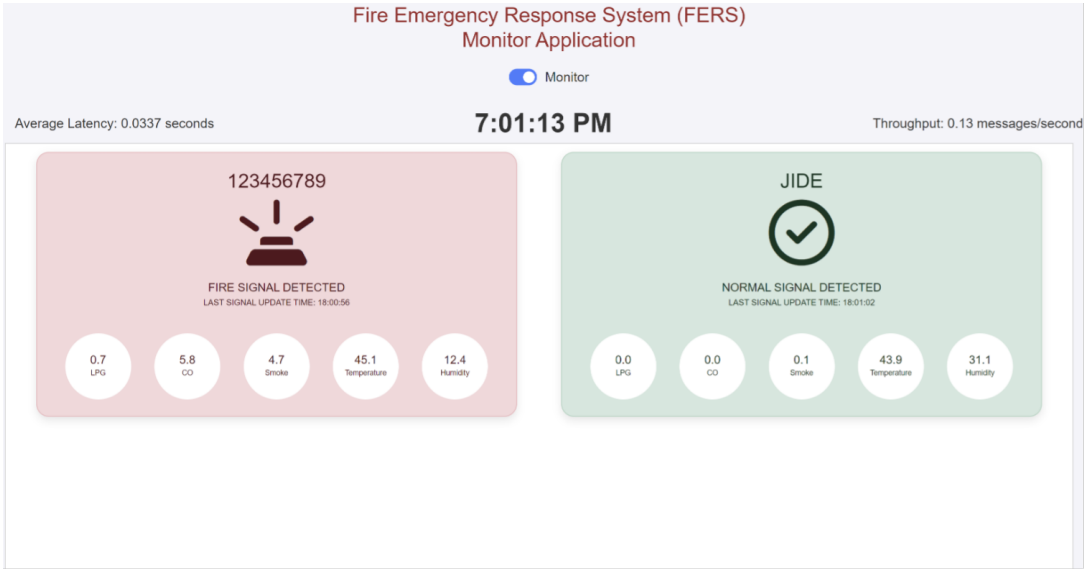

Figure 4: FERS Monitor Application when On

6.3 Discussion

The effectiveness of the Fire Emergency Response System (FERS) was evaluated using machine learning models and network capabilities. The machine learning models achieved 100% across all performance metrics. However, we observed that these results were obtained because the patterns in the dataset especially in the LPG, CO, and smoke values in fire signals were very distinguishable. This raises issues about how they are able to

make generalizations of real-world scenarios as datasets are often noisy and the patterns are less distinct which may cause overfitting. Network performance was evaluated based on latency and throughput, as these metrics are critical for real-time fire detection and response. We also observed that the system could deliver timely fire alerts and efficiently handle data transmission with very small delays. However, as more IoT sensor nodes are added, scalability should be put into consideration as it may stymied the system's ability to maintain low latency and high throughput.

The integration of machine learning, IoT sensors, and cloud computing in FERS creates a scalable and reliable fire detection system. However, the evaluation highlights areas where critical improvements can be made, including the need for further validation for a more realistic representation of real-world scenarios; and scalability testing for future expansion. By addressing these limitations, FERS can be further refined to ensure robust, reliable, and real-world-applicable fire detection and response capabilities.

7 Conclusion and Future Work

Summarily, the Fire Emergency Response System (FERS) clearly demonstrated a secure and sustainable solution for real-time fire detection through the combination of IoT sensors, machine learning algorithms, and cloud-based monitoring. Critical environmental drivers were tracked regularly by the DHT11 and MQ2 sensors, while MQTT protocols allowed for seamless data sharing with minimal lagging. The application that was used for monitoring was deployed on AWS and had an interface that was simple for the user to navigate as it displayed the system conditions, identifying variations with the normal operation states, fire alert and system off. As a result of the strong class distinctions of the datasets, the machine learning models reached target performance metrics as 100% accuracy, sensitivity and specificity were recorded, ensuring robust fire detection and timely interventions. Considering all factors, FERS not only achieved its research goals but also demonstrated its value for practical applications. In spite of its achievements, there are pitfalls in the FERS that require more research. For example, its effectiveness in hard-to-reach or connectivity -poor areas may be inhibited by the system's need for stable internet connection for integration. Additionally, the dataset's class distinctions, despite being beneficial for performance, failed to depict sufficiently the complicated nature of true fire situations, showing the need for a more extensive and diverse data set. Future research should direct attention to the integration of advanced computing technologies, expanding the dataset to boost the resilience and versatility of the machine learning models across different scenarios, and improving FERS for offline or hybrid functionality to ensure dependability in hard-to-reach areas.

References

Ahmed, K. and Gregory, M. (2011). Integrating wireless sensor networks with cloud computing, *Research Gate [Preprint]* .

Al-Dahoud, A. and et al. (2023). Forest fire detection system based on low-cost wireless sensor network and internet of things, *WSEAS TRANSACTIONS ON ENVIRONMENT AND DEVELOPMENT* **19**: 506–513.
URL: *https://doi.org/10.37394/232015.2023.19.49*

Alzubi, J., Nayyar, A. and Kumar, A. (2018). Machine learning from theory to algorithms: An overview, *Journal of Physics Conference Series* **1142**: 012012.
URL: *https://doi.org/10.1088/1742-6596/1142/1/012012*

Bal, D. et al. (2021). Wsn based advanced emergency fire alarm system using mqtt centric star topology, *2021 IEEE International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, Vol. 1, pp. 156–160.

Başţanlar, Y. and Özuysal, M. (2013). Introduction to machine learning, *Methods in Molecular Biology*, pp. 105–128.
URL: *https://doi.org/10.1007/978-1-62703-748-8*$_7$

Bouabdellah, K., Noureddine, H. and Larbi, S. (2013). Using wireless sensor networks for reliable forest fires detection, *Procedia Computer Science* **19**: 794–801.
URL: *https://doi.org/10.1016/j.procs.2013.06.104*

Brito, T. et al. (2020). Wireless sensor network for ignitions detection: An iot approach, *Electronics* **9**(6): 893.

Deschambault, O., Gherbi, A. and LeGare, C. (2017). Efficient implementation of the mqtt protocol for embedded systems, *Journal of Information Processing Systems* . [Preprint].

Dinculeană, D. and Cheng, X. (2019). Vulnerabilities and limitations of mqtt protocol used between iot devices, *Applied Sciences* **9**(5): 848.

Duobiene, S. et al. (2022). Development of wireless sensor network for environment monitoring and its implementation using ssail technology, *Sensors* **22**(14): 5343.

Dwivedi, R. and Kumar, R. (2018). Sensor cloud: Integrating wireless sensor networks with cloud computing, *2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON).*
URL: *https://doi.org/10.1109/upcon.2018.8597008*

El-Hosseini, M. and et al. (2020). A fire detection model based on power-aware scheduling for iot-sensors in smart cities with partial coverage, *Journal of Ambient Intelligence and Humanized Computing* **12**(2): 2629–2648.
URL: *https://doi.org/10.1007/s12652-020-02425-w*

Eser, A. (2024). Industrial fires statistics: Market data report 2024, `https://worldmetrics.org/industrial-fires-statistics/`. Accessed: October 21, 2024.

Grover, K. et al. (2019). Wsn-based system for forest fire detection and mitigation, *Lecture Notes on Multidisciplinary Industrial Engineering*, pp. 249–260.

Hamdani, S. and Sbeyti, H. (2019). A comparative study of coap and mqtt communication protocols, *2019 IEEE International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–5.

Hamzah, H. and Abdul-Rahaim, L. (2022). Design and implementation of fire alarm for big city based on cloud computing system, *IEEE [Preprint]* .
URL: *https://doi.org/10.1109/aest55805.2022.10412882*

Hu, Z. et al. (2024). Edge computing-based wildfire detection and optimization algorithm, *IEEE [Preprint]* .

Imran, M. A., Hussain, S. and Abbasi, Q. H. (2020). *Wireless Automation as an Enabler for the Next Industrial Revolution*, John Wiley Sons.

Kanoun, O. et al. (2021). Energy-aware system design for autonomous wireless sensor nodes: A comprehensive review, *Sensors* **21**(2): 548.

Karaduman, B. and et al. (2018). A cloud and contiki based fire detection system using multi-hop wireless sensor networks, *ACM*, pp. 1–5.
URL: *https://doi.org/10.1145/3234698.3234764*

Kizilkaya, B. and et al. (2022). An effective forest fire detection framework using heterogeneous wireless multimedia sensor networks, *ACM Transactions on Multimedia Computing Communications and Applications* **18**(2): 1–21.
URL: *https://doi.org/10.1145/3473037*

Manvi, S. S. and Shyam, G. K. (2013). Resource management for infrastructure as a service (iaas) in cloud computing: A survey, *Journal of Network and Computer Applications* **41**: 424–440.

Microsoft (n.d.). What is paas? platform as a service — microsoft azure, `https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas`. Accessed: January 24, 2025.

Modisane, P. and Jokonya, O. (2021). Evaluating the benefits of cloud computing in small, medium and micro-sized enterprises (smmes), *Procedia Computer Science* **181**: 784–792.

Ouda, G. K. and Yas, Q. M. (2021). Design of cloud computing for educational centers using private cloud computing: A case study, *Journal of Physics Conference Series* **1804**(1): 012119.

Sandhu, A. K. (2022). Big data with cloud computing: Discussions and challenges, *Big Data Mining and Analytics* **5**(1): 32–40.

Saputra, F., Rasyid, M. and Abiantoro, B. (2017). Prototype of early fire detection system for home monitoring based on wireless sensor network, *IEEE [Preprint]* .
URL: *https://doi.org/10.1109/elecsym.2017.8240373*

Sarwar, B. and et al. (2018). Design and application of fuzzy logic based fire monitoring and warning systems for smart buildings, *Symmetry* **10**(11): 615.
URL: *https://doi.org/10.3390/sym10110615*

Sharma, R., Rani, S. and Memon, I. (2020). A smart approach for fire prediction under uncertain conditions using machine learning, *Multimedia Tools and Applications* **79**(37–38): 28155–28168.
URL: *https://doi.org/10.1007/s11042-020-09347-x*

Sharma, S., Bansal, R. and Bansal, S. (2013). Issues and challenges in wireless sensor networks, *IEEE [Preprint]* .
URL: *https://doi.org/10.1109/icmira.2013.18*

Sowah, R. and et al. (2016). Design and implementation of a fire detection and control system for automobiles using fuzzy logic, *IEEE [Preprint]* .
URL: *https://doi.org/10.1109/ias.2016.7731880*

Sungheetha, A. and R, R. (2020). Real time monitoring and fire detection using internet of things and cloud based drones, *Journal of Soft Computing Paradigm* **2**(3): 168–174.
URL: *https://doi.org/10.36548/jscp.2020.3.004*

Varela, N. et al. (2020). Wireless sensor network for forest fire detection, *Procedia Computer Science* **175**: 435–440.

Wulf, F. et al. (2021). Iaas, paas, or saas? the why of cloud computing delivery model selection – vignettes on the post-adoption of cloud computing, *Proceedings of the Annual Hawaii International Conference on System Sciences*. [Preprint].

Yunhong, L. and Meini, Q. (2016). The design of building fire monitoring system based on zigbee-wifi networks, *IEEE [Preprint]* .
URL: *https://doi.org/10.1109/icmtma.2016.180*